TARGET TRACKING USING RESIDUAL VECTOR QUANTIZATION

A Dissertation
Presented to
The Academic Faculty

By

Salman Aslam

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy
in
Electrical and Computer Engineering

School of Electrical and Computer Engineering
Georgia Institute of Technology
December, 2011

TARGET TRACKING USING RESIDUAL VECTOR QUANTIZATION

Approved by:

Dr. David V. Anderson, Committee Chair
Assoc. Professor, School of ECE
Georgia Institute of Technology

Dr. Vijay Madisetti
Professor, School of ECE
Georgia Institute of Technology

Dr. Christopher F. Barnes, Advisor
Assoc. Professor, School of ECE
Georgia Institute of Technology

Dr. Patricio Vela
Assoc. Professor, School of ECE
Georgia Institute of Technology

Dr. Aaron F. Bobick, Co-advisor
Professor, School of Interactive Computing
Georgia Institute of Technology

Dr. Santanu Dey
Asst. Professor, School of ISYE
Georgia Institute of Technology

Date Approved: 11 November 2011

To my parents, my wife and my children.

ACKNOWLEDGMENTS

I would like to thank my advisor, Dr Christopher Barnes, for helping me in learning about a new and exciting method, residual vector quantization, my co-advisor, Dr Aaron Bobick, for his input and critiques on matters related to computer vision, and my committee for their feedback and guidance.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

In this work, our goal is to track visual targets using residual vector quantization (RVQ). We compare our results with principal components analysis (PCA) and tree structured vector quantization (TSVQ) based tracking.

This work is significant since PCA is commonly used in the Pattern Recognition, Machine Learning and Computer Vision communities. On the other hand, TSVQ is commonly used in the Signal Processing and data compression communities. RVQ with more than two stages has not received much attention due to the difficulty in producing stable designs. In this work, we bring together these different approaches into an integrated tracking framework and show that RVQ tracking performs best according to multiple criteria over a variety of publicly available datasets. Moreover, an advantage of our approach is a learning-based tracker that builds the target model while it tracks, thus avoiding the costly step of building target models prior to tracking.

CHAPTER 1

INTRODUCTION

1.1 Problem overview

Images have fascinated humans for thousands of years. The earliest cave paintings have been dated back to around 30,000 years [11]. Currently, several fields deal with the creation and analysis of images. Some example fields are art, photography, calligraphy and digital forensics. In this work, we are primarily interested in the computational processing of images. The three primary fields dealing with this aspect of images are Image Processing, Computer Vision and Computer Graphics. The field of Image Processing deals with low level image analysis, Computer Vision deals with high level image analysis, and Computer Graphics primarily deals with image synthesis. Within the field of Computer Vision, we are interested in tracking multiple objects in image sequences.

Object tracking, target tracking, or simply tracking, can be defined as estimating the trajectory of an object of interest over time. In practical applications, tracking is normally preceded by a detection step and succeeded by a track analysis step [2]:

- Detection. In this step, objects of interest are identified and segmented. A background model is commonly used as a pre-processing step.
- Tracking. The detected objects of interest are tracked from frame to frame.
- Track analysis. In this step, track information is fused to infer higher semantic knowledge.

Recently, there has been a surge in interest in tracking due to several reasons: (a) proliferation of powerful computers, (b) availability of high quality and inexpensive video cameras, and (c) an increasing need for automated video analysis. However, in practice, many difficulties are encountered in visual tracking. Some of these challenges include,

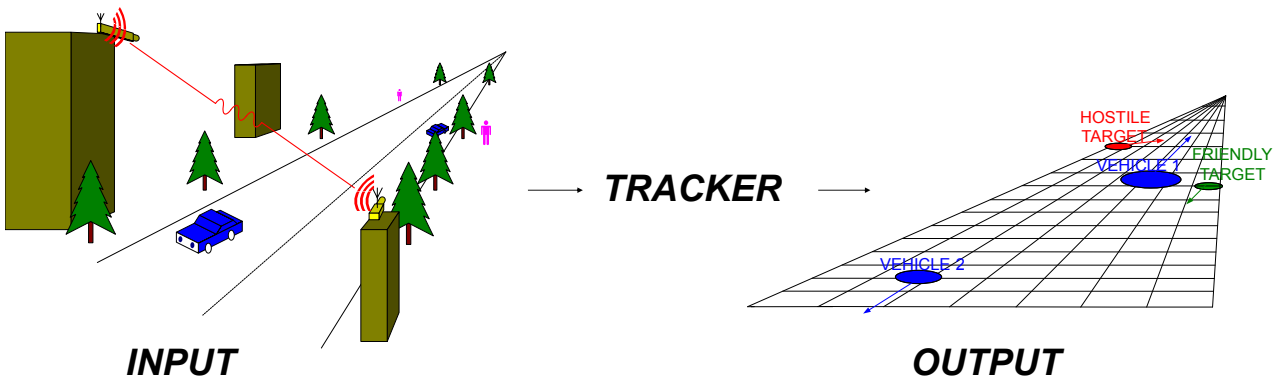


Figure 1. Illustration of a visual tracking scenario.

- Loss of information caused by projecting 3D world objects onto 2D images
- Sudden illumination changes
- Appearance drifts
- Complex target motion, including acceleration
- Non-rigid or articulated nature of objects
- Object-to-object merges/splits/occlusions
- Object-to-scene occlusions
- Camera motion
- Noise
- Real time processing requirements

Clearly, visual tracking is a challenging problem. Under general conditions, it remains an unsolved problem. Several researchers have tried to approach this problem by specifying additional constraints on the targets being tracked. Constraints have also been placed on the tracking environment. For example, almost all tracking algorithms assume that the object motion is smooth without any abrupt changes [2]. Furthermore, prior knowledge about the

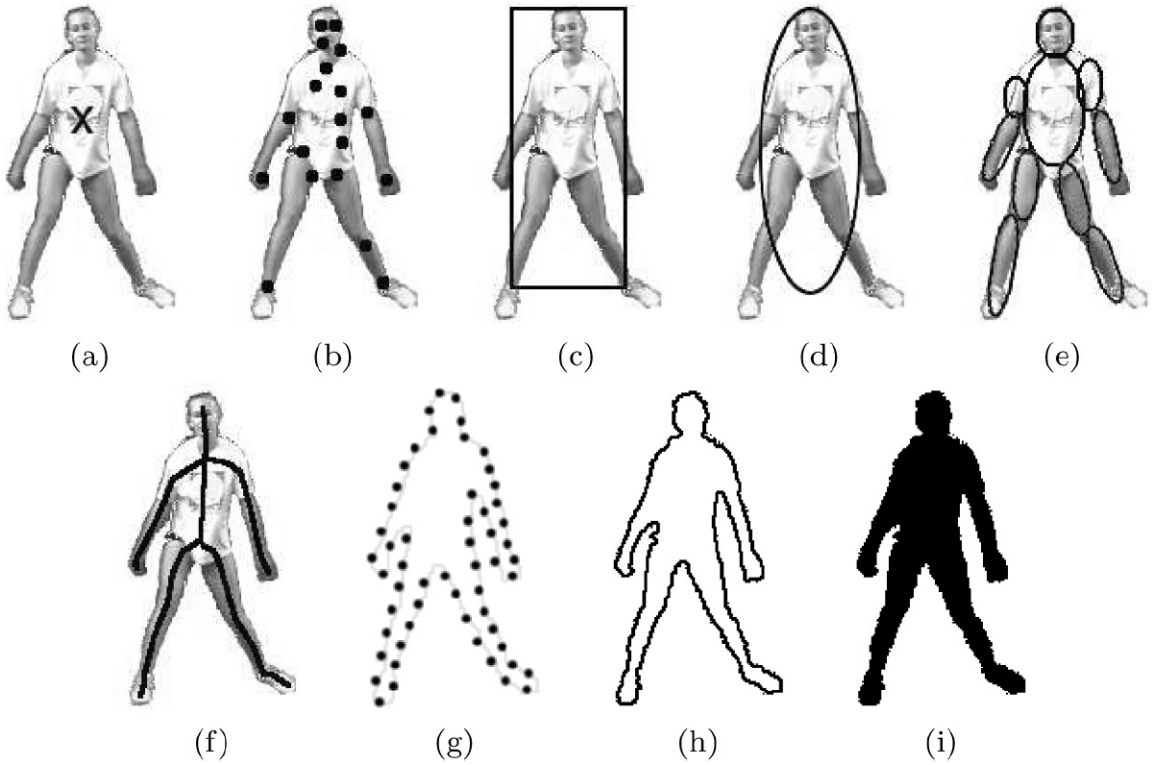


Figure 2. Target representations. (a) Centroid, (b) multiple points,(c) rectangular bounding box, (d) elliptical bounding region, (e) articulated shape model, (f) skeleton, (g) contour control points, (h) contour, (i) silhouette [2].

number, size, appearance and shape of the tracked objects has been used to simplify the problem.

A graphical illustration of the tracking problem is shown in Figure 1. In this figure, multiple cameras are employed for visual surveillance in an outdoor scenario. The images captured by these cameras are fed to a tracker which returns metadata about the objects being tracked, such as target ID and target velocity. Tracking is also commonly used in indoors applications, such as in tracking people in airports and malls. Single camera tracking is also still widely used although multi-camera tracking is an active area of research.

1.2 Solutions

In the preceding section, we outlined the tracking scenario and some challenges faced therein. We now turn to solutions.

1.2.1 Target representations

Several algorithms have been employed for single and multiple target tracking. The nature of the algorithm chosen is closely tied to the target representation. Several target representations are shown in Figure 2.

1.2.2 Tracking algorithms

The different target representations shown in Figure 2 lend themselves to the following general categories of trackers:

- Point tracker. Targets represented using centroids or multiple points are commonly tracked using point trackers. This form of tracking is closely tied to radar tracking. As a matter of fact, the same techniques used in radar tracking are used. Commonly used techniques include Kalman filtering, particle filtering [12], the *probabilistic data association filter* (PDAF) [13], the *joint probabilistic data association filter* (JPDAF) [14] and the *multiple hypothesis tracker* (MHT) [15].

- Region tracker. Targets represented using bounding boxes are commonly tracked using region tracking. Widely used tracking methods in this category are template matching and mean shift tracking [16].

- Contour tracker. Targets represented using shape information are commonly represented using splines and tracked using active contours [17] and level sets [18].

A closely associated problem is that of data association. An overview of data association methods in target tracking is given in [19]. More recently, particle filters have been shown to have an inherent capacity for data association [20].

1.2.3 Preprocessing

It is common to apply pre-processing steps before tracking is initiated. Some of these techniques include

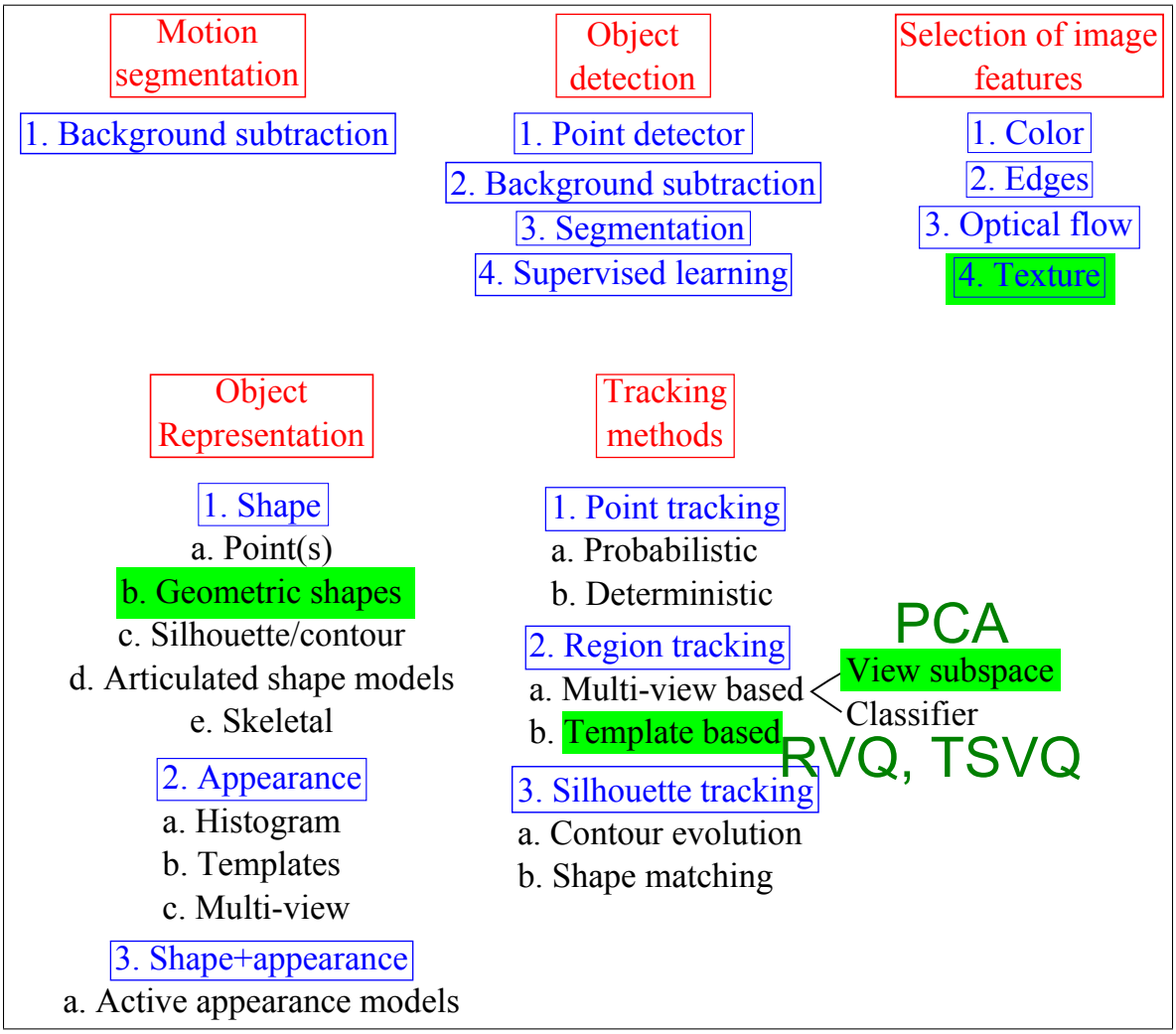


Figure 3. Visual tracking, pre-processing steps, object representations and tracking methods [2]. In this work, we use the *view subspace* method.

- Downsampling. This step is commonly carried out to reduce computational complexity.

- Normalization. This step is commonly used to normalize brightness variation in temporal sequences.

- Stabilization. Camera jitter is a common problem in tracking, especially in outdoor scenarios, or where cameras are hand-held. Different camera stabilization algorithms have been proposed to reduce the effect of camera motion. An overview can be found in [21, 22].

- Background modeling. Several methods have been suggested for background modeling. A commonly used method is the multi-Gaussian algorithm [23]. An overview of background modeling methods can be found in [24].

- Feature extraction. It is common to extract features from the target of interest and apply the algorithms mentioned above in the feature domain rather than in the raw spatial domain. Commonly used features include color, edges, corners, motion, texture, depth and spatial intensity probability distribution. For instance, corner detection [25, 26] can be used to extract points of interest within the target which are then tracked using point trackers.

An overview of these pre-processing steps, different object representations, and tracking methods is given in Figure 14.

1.3 Current work

In this work, we use a technique called Residual Vector Quantization (RVQ) for tracking. RVQ was first introduced in 1982 in the context of speech compression [27]. The first application of RVQ for image analysis appeared in 2007 where this method was used to analyze damage caused by Hurricane Katrina in the United States [28]. Our work presents the first usage of RVQ for any form of video analysis.

1.4 Outline

So far, we've briefly discussed three things in this introductory chapter: (a) the problem that we attempt to solve in this work, (b) the methods that have been employed in an attempt to solve this problem, and (c) the method that we intend to use in our own attempts. The remaining portion of this document elaborates on these issues. Here is an outline:

- Chapter 2. Residual Vector Quantization. In this chapter, we discuss RVQ, give the mathematical notation involved, and discuss design, optimality and implementation issues. A comparison with other vector quantization methods is also given.

- Chapter 3. Tracking methods. In this chapter, an overview of existing tracking methods is given.

- Chapter 4. RVQ tracking. In this chapter, visual tracking using RVQ is explained. Comparison is also made with tracking using two well known methods, *principal component analysis* (PCA) and *tree-structured vector quantization* (TSVQ). Six publicly available datasets are used that cover a variety of scenarios including indoors, outdoors, day-time, night-time, human, vehicle and object tracking, rigid and non-rigid object tracking, lighting change, structured noise, camera motion, target pose and expression changes, and temporary occlusions.

- Chapter 5. Results. In this chapter, results of PCA, TSVQ and RVQ based tracking are presented and compared.

- Chapter 6. Conclusions. This chapter wraps up this thesis briefly restating objectives, summarizing results, presenting conclusions and laying out a ground map for future work.

- Chapter 7. Appendices. This chapter includes detailed plots for the interested reader.

CHAPTER 2

RESIDUAL VECTOR QUANTIZATION (RVQ)

2.1 Introduction

Our main emphasis in this chapter is on Residual Vector Quantization (RVQ). However, in order to understand RVQ, certain definitions need to be presented. Also, an understanding of quantization, types of vector quantization and a comparison of different types of vector quantization is important. This chapter is therefore organized as follows:

1. Definitions. In Section 2.2, various definitions related to the study of quantization are presented.

2. Optimality In Section 2.3, we discuss optimality issues in quantization.

3. Types of vector quantization (VQ). In Section 2.4, we discuss different types of VQ, including exhaustive search vector quantization (ESVQ), tree structured vector quantization (TSVQ) and residual vector quantization (RVQ).

2.2 Definitions

In this section, we present the following definitions before getting into the details of optimality issues in quantization and its different types:

1. Quantization. Quantization is the process of representing a large, possibly infinite, set of values with a smaller set of values [29]. Figure 4 shows a quantizer Q that takes values from a source alphabet $\mathcal{X} = \{\mathbf{x} \in \mathbb{R}^D\}$ and maps them to a reconstruction alphabet $C = \{\mathbf{y}_k \in \mathbb{R}^D \mid k = 1, 2, \ldots K\}$. If the input is scalar, i.e. $D = 1$, the quantizer is called a *scalar quantizer*. For $D > 1$, the quantizer is called a *vector quantizer*. Quantization is used widely for achieving lossy compression in images

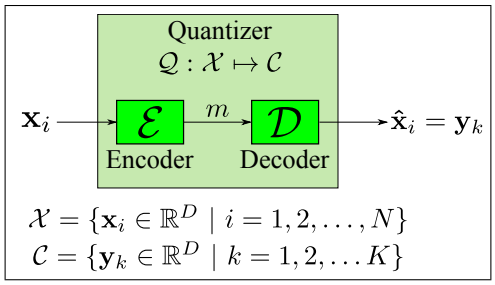


Figure 4. A quantizer Q maps symbols from a source alphabet $\mathcal{X}$ to symbols from a reconstruction alphabet $\mathcal{C}$, where in general, the number of elements in $\mathcal{X}$, $N >> K$, the number of elements in $\mathcal{C}$.

and videos. For instance, all current video standards, MPEG-1, MPEG-2, MPEG-4, H.261, H.263 and H.264 rely on a special form of quantization called *transform vector quantization*. See Figure 5 for an example of quantization in MPEG-4. In practice, quantization can be carried out as a sequence of two operations, *encoding* and *decoding*.

2. Encoding. During this process, the input $\mathbf{x}$ to be quantized is represented by an index m, usually a scalar, that corresponds to the code-vector $\mathbf{y}_k$ that $\mathbf{x}$ is mapped to.

3. Decoding. During this process, the index m is used to look up code-vector $\mathbf{y}_k$.

4. Parititions. Quantization creates K partitions $\mathcal{P}_k = \{\mathbf{x} \in \mathbb{R}^D \mid Q(\mathbf{x}) = \mathbf{y}_k\}$ in the input space $\mathbb{R}^D$ which are mutually exclusive and exhaustive, i.e., $\mathcal{P}_i \bigcap \mathcal{P}_j = \emptyset,\ i \neq j$. The union of these partitions covers the entire input space, $\bigcup_{k=1}^{K} \mathcal{P}_k = \mathbb{R}^D$.

5. Codebook. The reconstruction alphabet C is known as the *codebook*.

6. Code-vectors. The K members $\mathbf{y}_k$ of the reconstruction alphabet C are called *code-vectors*. The term *centroid* is used interchangeably with code-vector.

7. Design-time. In the context of quantization, design-time refers to the process of generating the codebook.

8. Run-time. In the context of quantization, run-time refers to the process of mapping an input $\mathbf{x}$ to a code-vector $\mathbf{y}$, i.e., the process of encoding followed by the process of

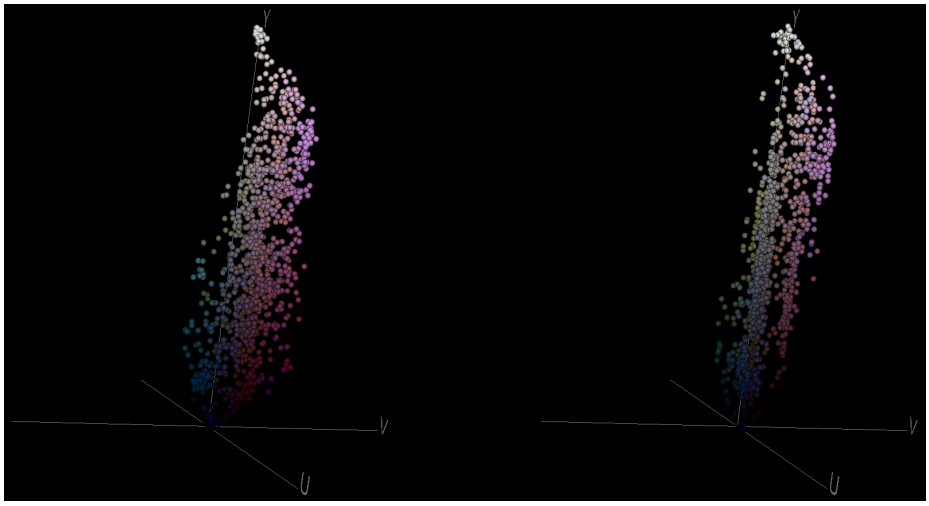


Figure 5. Scalar quantization in the transform domain for MPEG-4 Part2 Visual. The image on the left shows the 3 intensity channels of an input image patch drawn in the YUV color space. The vertical dimension is the luma (Y) axis. The right image shows the quantized reconstruction of the input image patch. No deblocking filter has been used, and so the loss of information is entirely due to quantization. Notice the straight lines along which the output pixels are aligned due to the quantization process. The visualization was created using the Visualization Toolkit (VTK) [3] in C.

decoding.

9. Rate. If we have K code-vectors $\mathbf{y}_k$ in $\mathbb{R}^D$, $\log_2 K$ bits are required to represent each code-vector. The *resolution*, *code rate*, or simply the *rate* r of a quantizer is the number of bits required to represent each sample, i.e., scalar element of $\mathbf{y}_k$. Since there are D samples, the rate $r = \frac{\log_2 K}{D}$.

10. Distortion. The difference between original input $\mathbf{x}$ and reconstructed output $\hat{\mathbf{x}} = Q(\mathbf{x}) = \mathbf{y}$ is known as *distortion* $d(\mathbf{x}, \mathbf{y})$ [29]. A commonly used distortion measure is the squared error criterion, $d(\mathbf{x}, \mathbf{y}) = (\mathbf{x}-\mathbf{y})^2$. Average distortion is given by $e(\mathcal{X}, C) = \mathrm{E}[d(\mathbf{x}, \mathbf{y})]$.

11. Rate-distortion R(D) curve. The tradeoff between rate and distortion can be plotted as a *rate-distortion* curve. A fundamental result of Shannon's rate-distortion theory is that VQ is able to achieve equal or better compression rates than scalar quantization even if the source is memoryless, i.e., emits a sequence of IID random variables [30]. The reason is that VQ is able to take advantage of higher dimensionality using appropriate cell-shapes [31]. For instance, it is shown in [31] that a hexagon quantizer

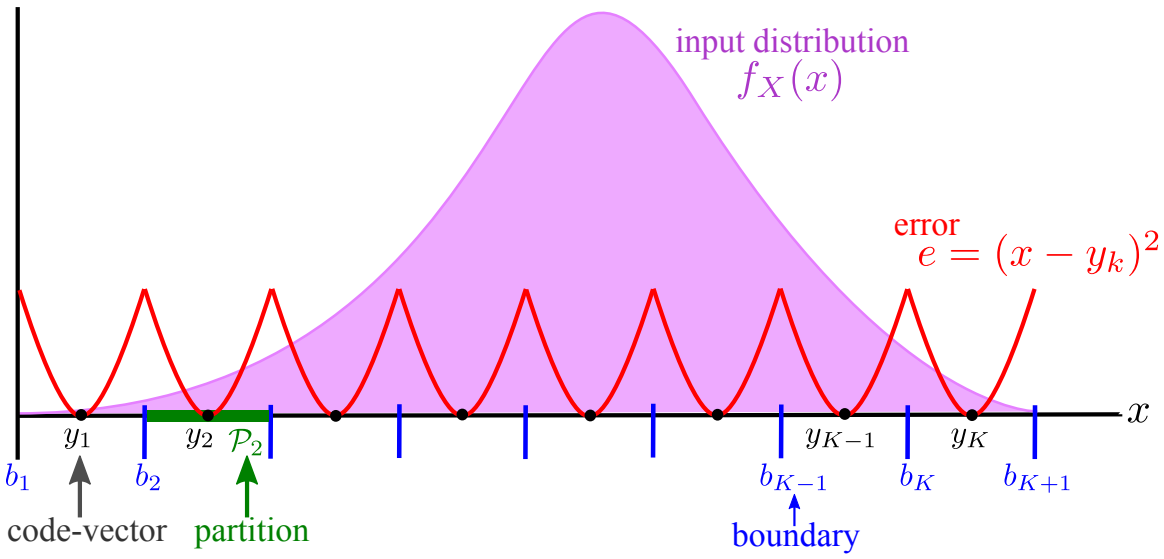


Figure 6. Given partition $\mathcal{P}_k$, the optimal code-vector for this partition is the centroid of the partition.

will gain 0.028 bits over a square quantizer in $\mathbb{R}^2$ when the two random variables are independent. On the other hand, if the source is not memoryless, 3 situations are possible:

- Linear correlation only. With proper rotation, any set of random variables can be rendered uncorrelated, i.e., they will no longer be linearly correlated [31]. In this case, since there is no non-linear correlation, the set of random variables after appropriate rotation will be independent. Scalar quantization along each new dimension will produce lower bit rate than if the rotation had not been carried out. However, VQ will reduce the bit-rate even more than scalar quantization since, as mentioned above, it can take advantage of appropriate cell-shapes.

- Non-linear correlation only. Here, rotation cannot be used to remove non-linear correlation and therefore scalar quantization on rotated axes will not improve bit rate. Moreover, scalar quantization produces rectangular cells, irrespective of the input distribution. VQ on the other hand is able to place centroids only in regions occupied by the input. This property can be used to exploit non-linear correlations and lower bit-rate [31]. Note that this property is independent of

the appropriate cell-shapes property. Using the appropriate cell-shapes property will lead to further reduction in bit-rate.

- Linear and non-linear correlation. In this case, VQ can be used to reduce bit rates using all 3 methods mentioned above, rotation, placement of code-vectors in places where the input distribution exists, and using appropriate cell-shapes.

12. Sigma (σ) tree. A σ-tree is shown in Figure 9. Each node of this tree, $\mu_{m,p}$ is called a *stage code-vector* and is the m-th node at the p-th stage. In Figure 9, there are 6 stage-code-vectors, 2 at the first stage, 2 at the second stage, and 3 at the third stage. The leaf nodes of this tree, also called *equivalent code-vectors* [5] constitute the RVQ code-book [32]. Each equivalent code-vector is created using a *direct sum*, i.e., by adding one stage code-vector from each stage. There are $K = M^P$ possible unique direct sums, and therefore $K = M^P$ possible equivalent code-vectors.

2.3 Quantization optimality

So far, no mention has been made about optimality of the code-vectors or partitions. A widely used algorithm to compute at least locally optimal codevectors and partitions is the Generalized Lloyd Algorithm (GLA) [4], also known as the Linde Buzo Gray (LBG) algorithm [33] or K-means clustering [34]. Figure 6 illustrates the scalar quantization case for an input X with distribution $f_X(x)$ and average distortion e,

$$\begin{aligned} e &= \int_{-\infty}^{\infty} (x - \hat{x})^2 f_X(x) dx \\ &= \sum_{k=1}^{K} \int_{b_k}^{b_{k+1}} (x - y_k)^2 f_X(x) dx \end{aligned} \tag{1}$$

If the partition $\mathcal{P}_k = \left\{ x \mid (y_k - x)^2 < (y_j - x)^2,\ \forall\ j \neq k \right\}$ is given, the optimal code-vector y_k for this partition can be computed by setting the derivative of the average distortion e with respect to y_k equal to 0, i.e., $\frac{\partial e}{\partial y_k} = 0$. Solving for y_k, we get,

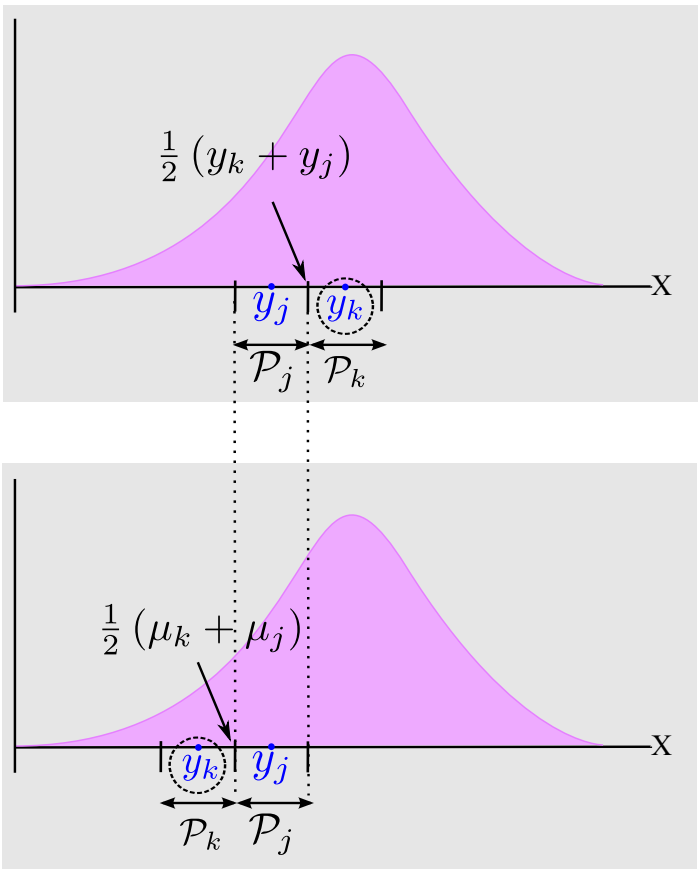


Figure 7. Given optimal centroids y_j and y_k, the optimal partition boundary is half-way between them.

$$y_k = \frac{\int_{b_k}^{b_{k+1}} x f_X(x) dx}{\int_{b_k}^{b_{k+1}} f_X(x) dx} \tag{2}$$

In other words, the optimal centroid for a given partition and the squared error criterion is the centroid of the partition. Conversely, if we want to compute optimal partitions given the centroids, we can rewrite $\mathcal{P}_j = \left\{x \mid (y_j - x)^2 < (y_k - x)^2,\ \forall\ j \neq k\right\}$ as

$$P_j = \left\{x \mid (y_k - y_j)\left(x - \frac{1}{2}\left(y_k + y_j\right)\right) < 0,\ \forall j \neq k\right\} \tag{3}$$

For the scalar case in Figure 7, if $k > j$, i.e., $(y_k - y_j) > 0$, then to satisfy Equation 3,

$$x - \frac{1}{2}\left(y_k + y_j\right) < 0$$
$$\Rightarrow x < \frac{1}{2}\left(y_k + y_j\right) \tag{4}$$

Conversely, if $k < j$, i.e., $(y_k - y_j) < 0$, then to satisfy Equation 3

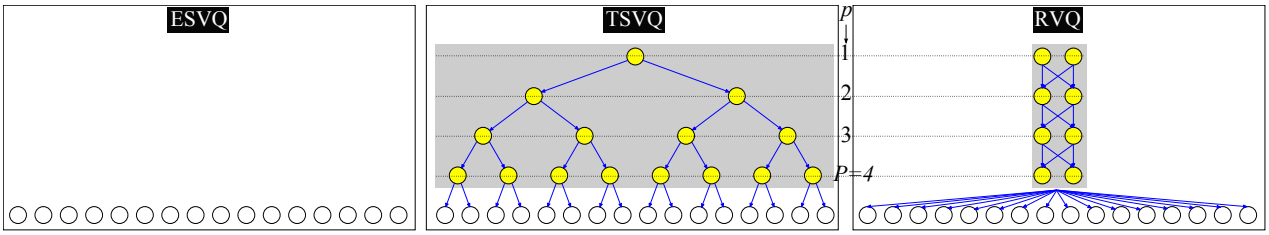
$$x - \frac{1}{2}\left(y_k + y_j\right) > 0$$
$$\Rightarrow x > \frac{1}{2}\left(y_k + y_j\right) \tag{5}$$

The only way to satisfy both Equations 4 and 5 is for the partition boundary to be halfway between the centroids, i.e., $\frac{1}{2}\left(y_k + y_j\right)$. This notion can be generalized to the vector case.

2.4 Types of VQ

We now list the main types of VQ that appear in the literature. The goal of VQ design is to have output distortion as close as possible to the rate-distortion curve. However, in general, optimal coding of source vectors is not possible unless an exhaustive search over all code-vectors is carried out, as in structurally unconstrained *Exhaustive Search Vector Quantizers* (ESVQs) [35]. For a rate r and dimension D, there are $K = 2^{rD}$ code-vectors. Therefore, the computational cost of ESVQ, C_{ESVQ}, and memory requirements M_{ESVQ} are $\approx 2^{rD}$. A solution to this problem is to impose constraints on the VQ structure.

One possible solution is the tree structured vector quantizer (TSVQ) proposed in [36]. A P-level binary TSVQ has run-time search complexity which is only $C_{TSVQ} \approx 2P$ but double storage requirements, $M_{TSVQ} \approx 2M_{ESVQ}$ [37]. So, although $TVSQ$ solves the search complexity problem, it further aggravates the storage problem. A method of reducing both run-time computational and storage complexity is to use a product code VQ [4]. The basic idea in a product code VQ is to break a bigger problem into several smaller problems. Examples include mean-residual VQ, gain-shape VQ and mean-gain-shape VQ [37]. Residual



		ESVQ	TSVQ	RVQ
1	# data points	N	same	same
2	Vertical dimension, P	-	depth	# stages
3	Horizontal dimension, M	-	breadth	# code-vectors per stage
4	Encoding indeces	-	path map (P-tuple)	XDR (P-tuple)
5	Input dimension	D	same	same
6	Rate, r (bits/vector component)	$(\log_2 K)/D$	same	same
7	# non-terminal (stage) nodes, K_n	0	$1 + M + M^2 + \ldots + M^{P-1} = \frac{M^P-1}{M-1}$	-
8	# terminal code-vectors, K	2^{rD}	M^P	same as TSVQ
9	search complexity/stage	1 stage, $O(2^{rD})$	$O(M)$	same as TSVQ
10	search complexity, all stages	$O(2^{rD})$	$O(MP)$	same as TSVQ
11	Total code-vectors to store in memory	2^{rD}	$MK_n = M\frac{M^P-1}{M-1}$	MP

Figure 8. Comparison of ESVQ, TSVQ and RVQ. In the top figure, $M = 2$ for RVQ and TSVQ, and 16 code-vectors are displayed for each quantization type. The term *path map* is used in [4] to denote the P encoding indeces. An equivalent term for RVQ, *expanded digital representation* (XDR) is used in [5].

Vector Quantizers (RVQ) also fall under this category, and are of interest to us in this work. A comparison of ESVQ, TSVQ and RVQ is given in Figure 8.

2.4.1 TSVQ

The Tree Structured Vector Quantizer (TSVQ) has received a lot of attention in the literature [4]. The reason is that the codebook produced by TSVQ approximates the codebook produced by ESVQ but the run-time computational cost is logarithmic in the number of code-vectors. For instance, a codebook size of $K = 256$ requires 256 matches for ESVQ but only 8 matches for binary TSVQ. However, as mentioned earlier, the storage requirements are greater for TSVQ as compared to ESVQ (see Figure 8). We next talk about design-time and run-time in TSVQ.

1. Design-time. The goal here is to design the TSVQ codebook which comprises the terminal code-vectors, i.e., the leaf nodes, in the TSVQ tree. The first step is to compute the mean of the training data. The mean is then split off into M_{TSVQ} child

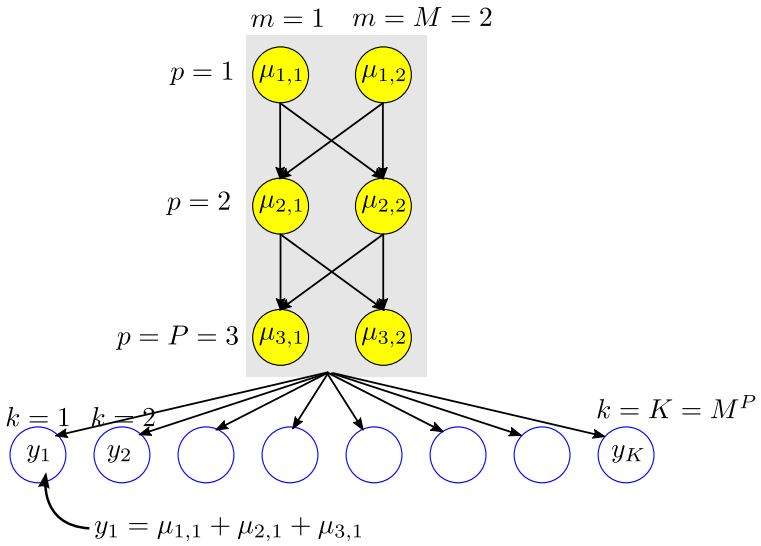


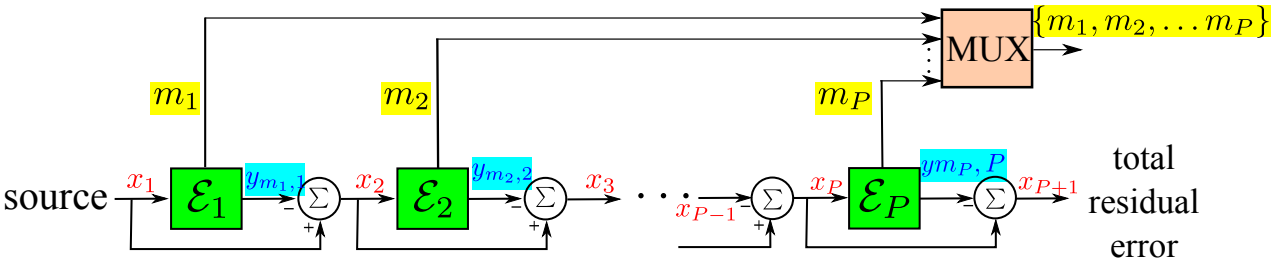
Figure 9. RVQ σ-tree, 3 stages, 2 code-vectors per stage, i.e., P=3, $M = 2$. This is a 3x2 σ-tree.

centroids (code-vectors). The training data is then mapped to these child centroids using the nearest neighbor rule. Each of these M_{TSVQ} child centroids are then again split into M_{TSVQ} centroids and the process continues until terminal code-vectors are obtained.

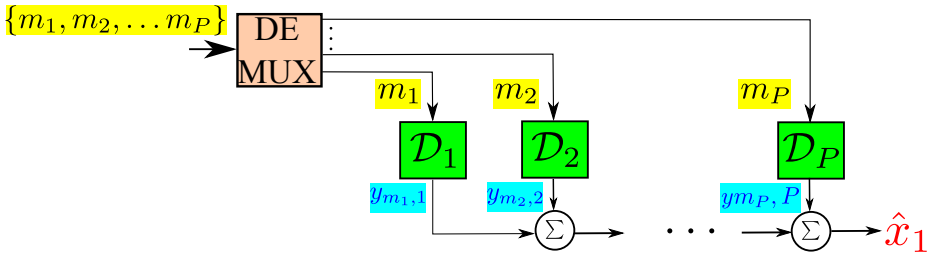
2. Run-time. During run-time, the encoding process involves mapping a test vector to the nearest centroid at each level of the tree. The index of the terminal code-vector is used to decode the test vector if no successive approximation is required. Note that in this case, only the terminal code-vectors need to be stored at the decoder. However, if successive approximation behavior is desired, then the entire tree needs to be stored and the stage map is used to successively approximate the input vector [4].

2.4.2 RVQ

Residual Vector Quantizers were introduced by Juang et al. [27] in 1982. As with ESVQ and TSVQ, the K-means, or GLA, objective function to be minimized for RVQ for the discrete case can be written as,



(a) Encoder.



(b) Decoder.

Figure 10. RVQ encoding and decoding.

$$e = \sum_{k=1}^{K} \sum_{\substack{i=1 \\ x_i \in C_k}}^{N} (x_i - y_k)^2 \tag{6}$$

Notice that in this equation, it is implicit that the partitions, i.e., Voronoi regions, are known. Computing both optimal partitions and optimal centroids is an NP hard problem. However, once the partitions are known, computing the optimal centroids is a convex least squares problem and can be solved by setting the derivative of the objective function with respect to the required code-vector equal to 0. As in the continuous case mentioned earlier, the optimal code-vectors are the centroids of the Vornoi regions.

For RVQ, the k-th equivalent code-vector is a direct sum of P stage code-vectors,

$$y_k = \mu_1^{(k)} + \mu_2^{(k)} + \ldots + \mu_P^{(k)} \tag{7}$$

Substituting this notation in Equation 6 and grouping all stage code-vectors except for the stage code-vector at the ρ-th stage gives us a series of equivalent equations, one equation per stage,

$$
\begin{aligned}
e &= \sum_{k=1}^{K} \sum_{\substack{i=1 \\ x_i \in C_k}}^{N} \left[x_i - \left(\sum_{p=2}^{P} \mu_p^{(k)} + \mu_\rho^{(k)} \right) \right]^2, \quad \rho = 1 \\
&= \sum_{k=1}^{K} \sum_{\substack{i=1 \\ x_i \in C_k}}^{N} \left[x_i - \left(\sum_{\substack{p=1 \\ p \neq 2}}^{P} \mu_p^{(k)} + \mu_\rho^{(k)} \right) \right]^2, \quad \rho = 2 \\
&\vdots \\
&= \sum_{k=1}^{K} \sum_{\substack{i=1 \\ x_i \in C_k}}^{N} \left[x_i - \left(\sum_{p=1}^{P-1} \mu_p^{(k)} + \mu_\rho^{(k)} \right) \right]^2, \quad \rho = P
\end{aligned} \tag{8}
$$

Equation 8 can be regrouped and written in compact notation as,

$$
\begin{aligned}
e &= \sum_{k=1}^{K} \sum_{\substack{i=1 \\ x_i \in C_k}}^{N} \left[\left(x_i - \sum_{\substack{p=1 \\ p \neq \rho}}^{P} \mu_p^{(k)} \right) - \mu_\rho^{(k)} \right]^2, \quad \rho = \{1, 2, \ldots P\} \\
&= \sum_{k=1}^{K} \sum_{\substack{i=1 \\ g_i \in \mathcal{H}_k}}^{N} (g_i - \mu_\rho^{(k)})^2, \quad \rho = \{1, 2, \ldots P\}
\end{aligned} \tag{9}
$$

where g_i is the *graft residual* [38]. As can be seen in Equation 9, the graft residual g_i for a data-point x_i is formed by subtracting from x_i, all stage codevectors that are used to reconstruct x_i except the stage codevector at the ρ-th stage. In this sense, g_i is a causal anti-causal (CAC) residual [38]. The code-vectors at the ρ-th stage are computed using the K-means objective function for that particular stage. The implication of this step is that the RVQ objective function is now a coupled K-means objective function where the design of each stage code-vector depends on stage code-vectors from all other stages, and not just prior stages, hence the name causal anti-causal. A challenge in this coupled K-means setup is that computing the centroids for one stage changes the residual centroids for all other stages.

An RVQ is different from a traditional VQ in the sense that it partitions the input space $\mathbb{R}^D$ into M cells. The residual space, also in $\mathbb{R}^D$, is then partitioned again into M cells. This process is repeated P times. The advantage of this approach is that in obtaining M^P

partitions, we need to run our partitioning algorithm P times and generate M partitions at each stage. In traditional VQ, the partitioning algorithm would run once but have to create M^P partitions. For the binary case (two code-vectors per stage, $M = 2$) and a total of 8 stages (P=8), RVQ only requires 16 searches. In $ESVQ$, this would require 256. The exponential complexity is reduced to linear complexity. In general, structurally constrained quantizers cannot provide performance as good as ESVQ. However, since they are able to more efficiently implement codes, larger and larger vector sizes can be used, and if carefully designed, can achieve better performance that ESVQ for a given computational cost [37].

Below, we give one method of designing an RVQ codebook. Refer to [39, 35, 40, 41, 38, 42, 43, 37, 32, 44, 28, 5] for details on other methods as well as a discussion on sequential and joint optimality.

1. Design-time. The following steps are taken:

   (a) Generating the σ-tree. The K-means algorithm is used to design first stage code-vectors. This is a standard application of the K-means algorithm as in ESVQ or in TSVQ. First stage code-vectors are then subtracted from the data points that map to them to generate a set of residual data points. The K-means algorithm is run on these residual data points and a set of second stage code-vectors is obtained. This process is repeated till the desired number of P stages.

   (b) Applying the CAC condition. The above step generates cluster centroids that are locally optimal at every stage. Morever, the design of each stage depends on the previous or causal stage designs but does not depend on subsequent or anti-casual stage designs. This can lead to a propagation of reconstruction error. However, as mentioned earlier, we would like to compute stage code-vectors using not just causal residuals, but causal and anti-causal residuals. For this, Equation 9 is repeatedly applied to all stages as shown in Figure 11.

2. Run-time. The RVQ run-time process is shown in Figure 12. At the first stage, the

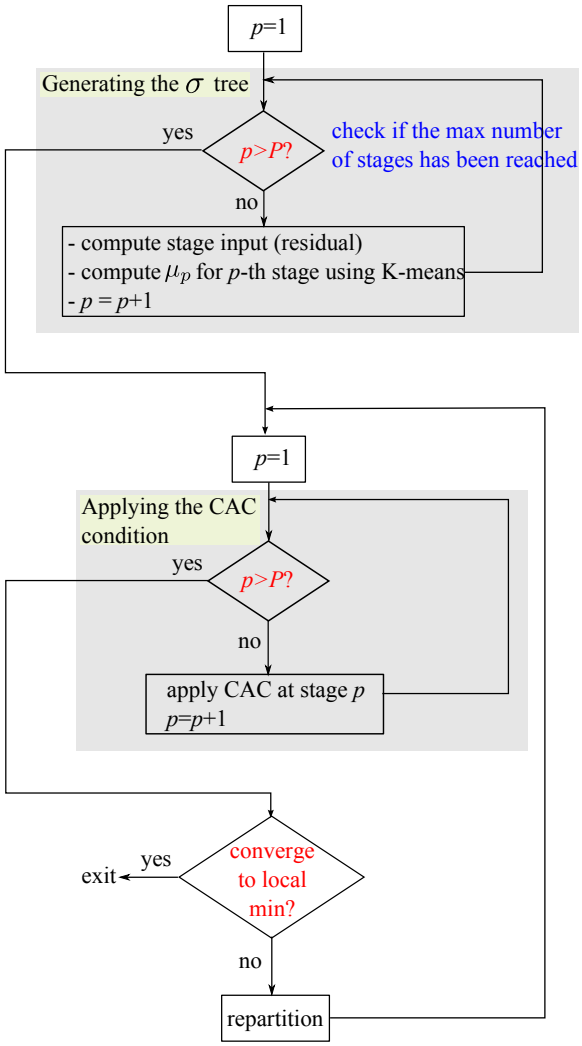


Figure 11. RVQ, design-time.

stage codevector with the least L_2 norm error is picked. This codevector is subtracted from the input signal to form a first stage residual signal. This signal is fed as input to the second stage where again the best second stage codevector in the L_2 norm sense is picked. The residual from this stage is fed as input to the third stage. This process is repeated for all P stages. The final residual output from the P-th stage is the error signal. The reconstructed output signal is a sum of the selected stage code-vectors at every stage. The computational complexity of this process is linear in the number of data-points, i.e., $O(N)$.

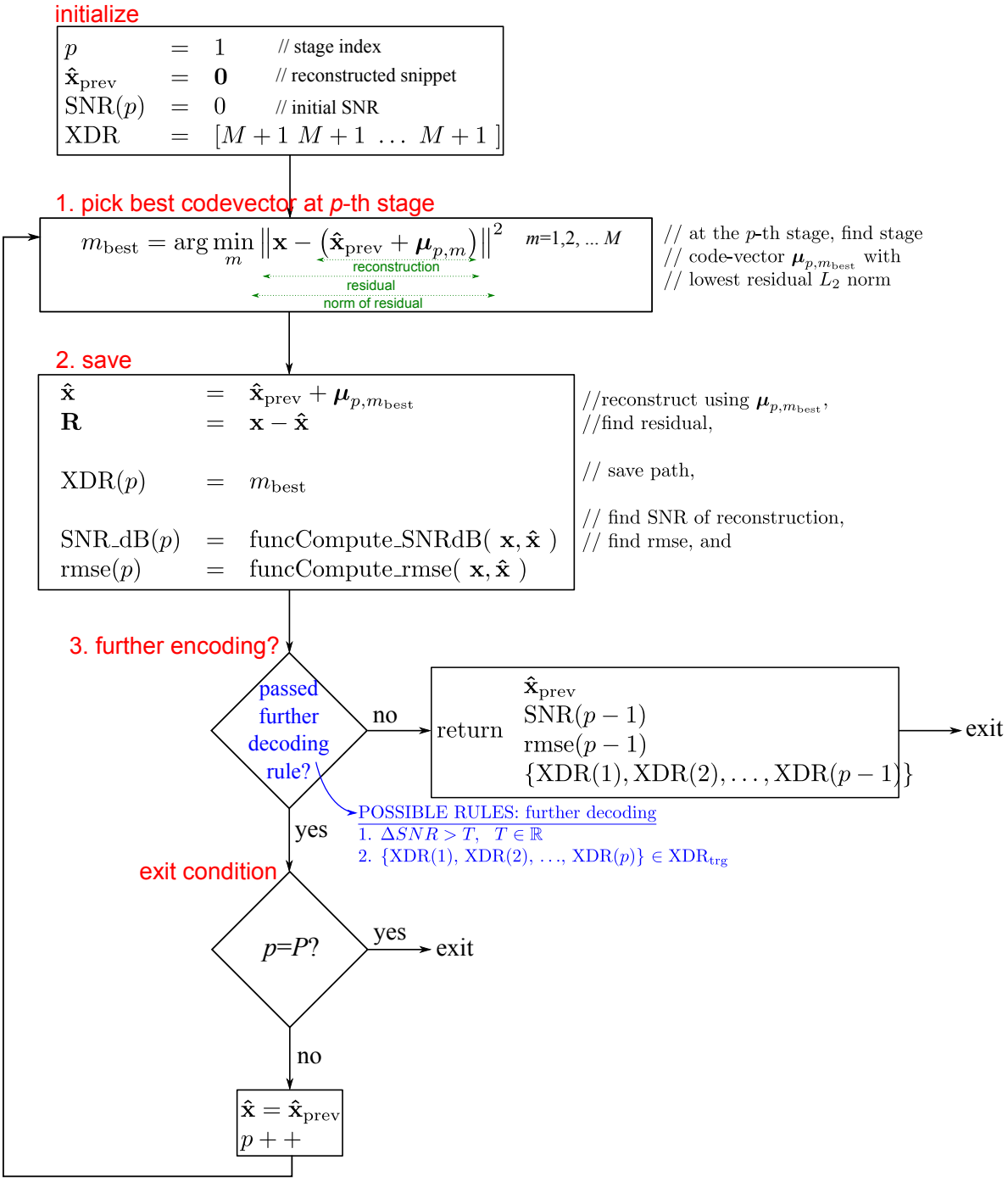


Figure 12. RVQ, run-time.

CHAPTER 3

TRACKING METHODS

In this chapter, we provide an overview of tracking methods that have been reported in the literature. The outline of this chapter is as follows,

- Bayesian estimation. Bayesian estimation is a general framework for target state estimation, and has been applied to the three major categories of tracking, point tracking, region tracking and contour tracking. We use this approach in this work as well. Therefore, this chapter begins with providing an overview of this method.

- Point tracking. Point tracking is the first of three tracking categories. The methods used in computer vision are very similar to the methods in the radar tracking literature.

- Region tracking. Region tracking is the second tracking category. In this method, entire regions are tracked. In this work, we use a sub-category of this approach called *subspace tracking*.

- Contour tracking. Contour tracking is the third and last tracking category. In this method, target contours are tracked.

- Example trackers. After discussing the above topics, we provide examples of some trackers that have received attention in the literature.

- Subspace tracking. Finally, we discuss subspace tracking, a form of region tracking. This section leads up to the approach we have used in this work, PCA, TSVQ and RVQ based tracking.

As mentioned earlier, visual tracking is an important and difficult area of computer vision and has received a lot of attention in the literature. Refer to [45, 2, 46] for surveys on this topic.

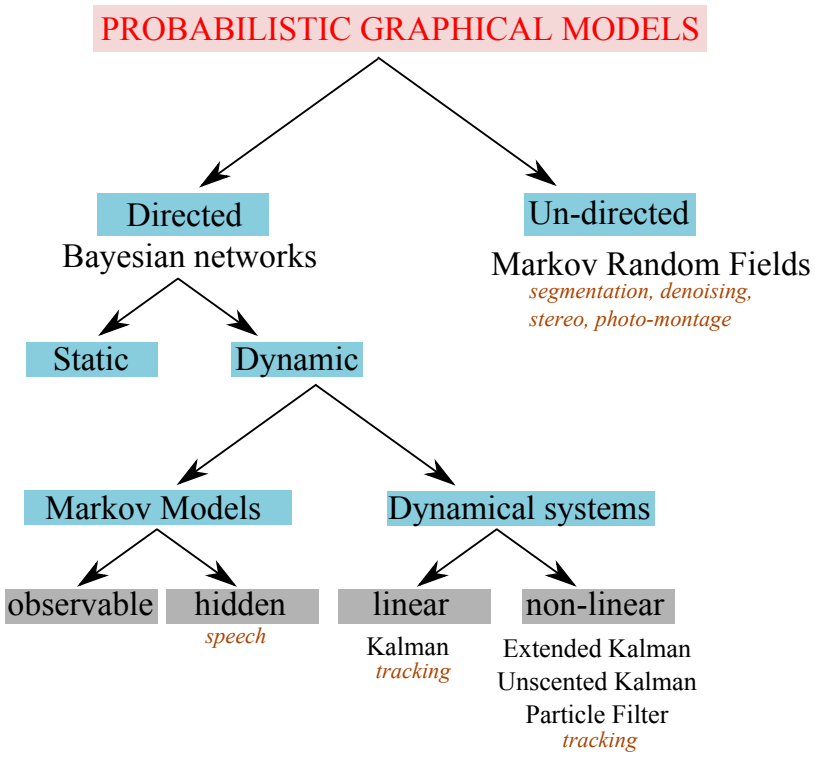


Figure 13. The process of tracking within the probabilistic graphical model hierarchy.

The biggest challenge in tracking is difficulty in handling changes in target appearance [1]. Intrinsic variations include pose variation and shape deformations. Extrinsic variations include illumination change, camera motion, camera viewpoint and occlusions. In order to simplify the process of tracking, it is common to make certain assumptions. These assumptions fall under two broad categories:

- Assumptions related to camera. A common assumption, in particular in surveillance applications, is a stationary camera that allows for background maintenance. Known camera-motion is also used although it is less common.

- Assumptions related to target. These assumptions include constant velocity, constant acceleration, coherent motion (all parts of the target move together) and motion along a straight path.

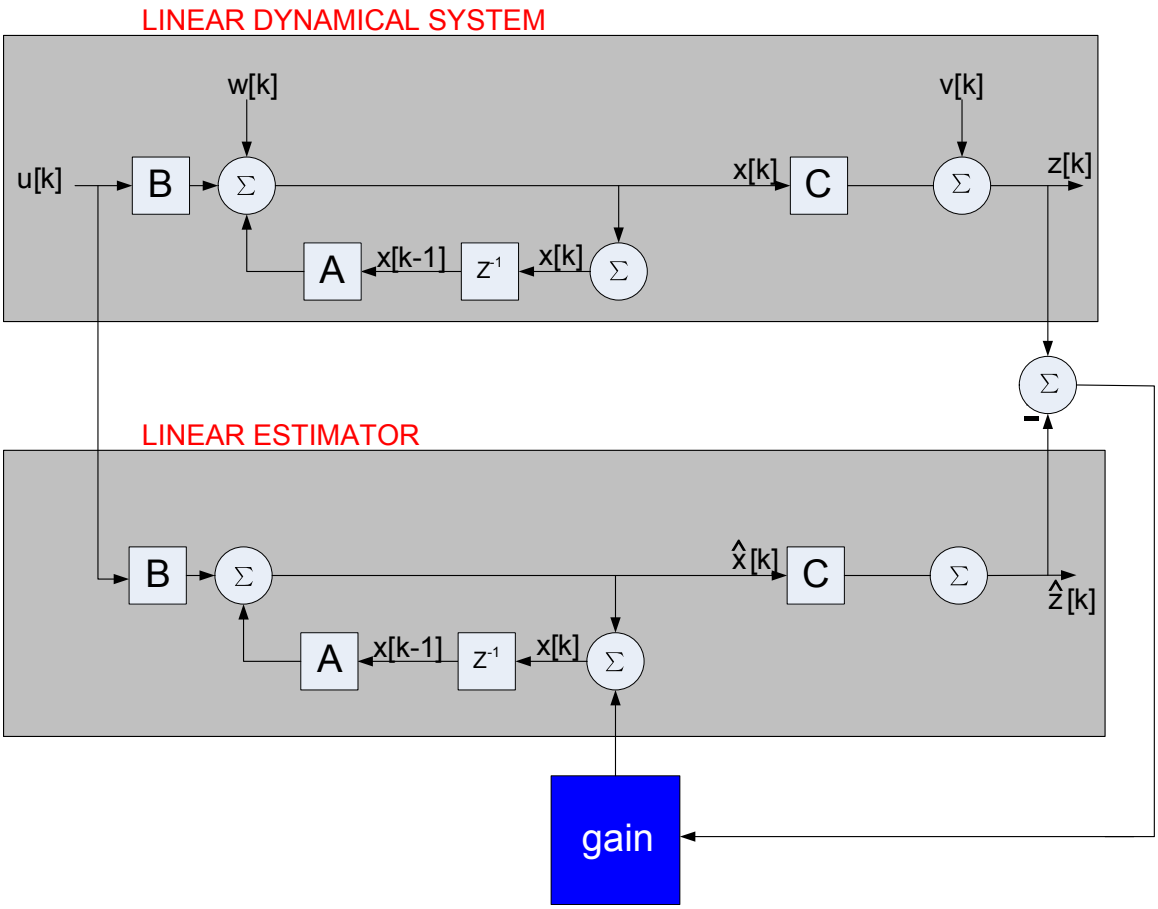


Figure 14. Linear estimator. The gain block is called the *Kalman gain* for the Kalman filter.

As stated previously, several pre-processing steps may be required before tracking can be initiated. These include stabilization for video registration, normalization, downsampling, background modeling and feature extraction. Commonly used features include raw pixel values, corners, area, color information, intensity distributions, contour descriptors and depth. In this work, the only pre-processing step we carry out is feature extraction of raw pixels that are expected to belong to the target of interest.

3.1 Bayesian estimation

A commonly used formulation for tracking is based on Bayesian estimation, and is used in this work as well. In this framework, target kinematics are modeled as the latent states of a

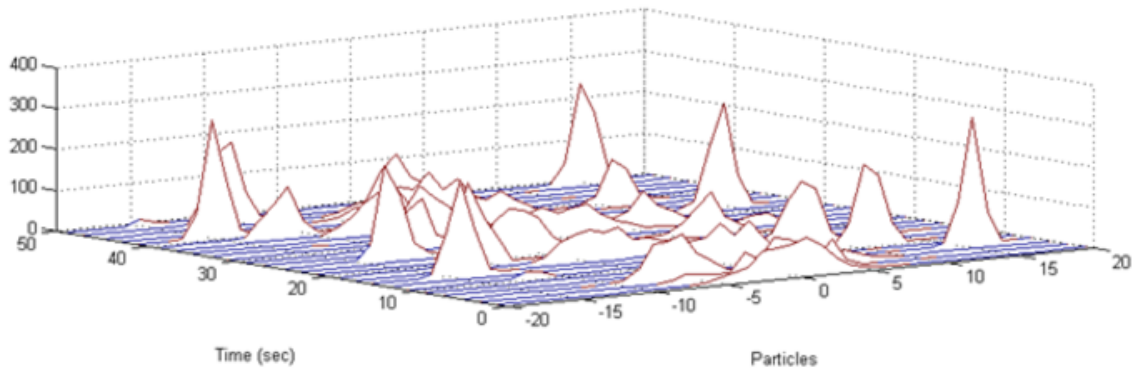


Figure 15. Tracking using a particle filter. Notice that the density is non-Gaussian and multi-modal.

time-dynamic system [12]. Time-dynamic systems are based on two models: (a) *state prediction model*, $f_t : \mathbb{R}^D \times \mathbb{R}^D \rightarrow \mathbb{R}^D$, describing state evolution, and (b) *observation model*, $h_t : \mathbb{R}^N \times \mathbb{R}^N \rightarrow \mathbb{R}^N$, relating observations to the states. These models are described as,

$$\begin{aligned} \mathbf{x}_t &= f_t(\mathbf{x}_{t-1}, \mathbf{v}_{t-1}) \\ \mathbf{z}_t &= h_t(\mathbf{x}_t, \mathbf{n}_t) \end{aligned} \tag{10}$$

$\mathbf{v} \in \mathbb{R}^D$ is an independent, identically-distributed (IID) process noise sequence. $\mathbf{n} \in \mathbb{R}^N$ is an IID measurement noise sequence. The goal is to find the estimate of the state $\mathbf{x}_t$ at time t, based on all observations $\mathbf{Z}_t = \{\mathbf{z}_i, i = 1, ..., T\}$. $\mathbf{z}_t$ is the observation vector at time t.

At this point, it is interesting to place the process of tracking in the bigger picture of probabilistic graphical models, as shown in Figure 13. Mathematically, hidden Markov models (HMMs) can also be written using evolution and observation models even though the method was developed independently of time dynamic systems [47].

The two stage time-dynamic model described above lends itself well to Bayesian inference [12]. The reason is that observations can be used as evidence to modulate the prior distribution on the states. We can then infer the posterior distribution on the states using Bayes' Rule. Mathematically, the Chapman Kolmogorov equation predicts the next state by combining information from the state prediction model $p(\mathbf{x}_t|\mathbf{x}_{t-1})$ and all previous observations $\mathbf{Z}_{t-1}$.

$$
\begin{aligned}
p(x_t|Z_{t-1}) &= \frac{p(x_t,Z_{t-1})}{p(Z_{t-1})} \\
&= \frac{\int p(x_t,x_{t-1},Z_{t-1})dx_{t-1}}{p(Z_{t-1})} \\
&= \frac{\int p(x_t|x_{t-1},Z_{t-1})p(x_{t-1},Z_{t-1})dx_{t-1}}{p(Z_{t-1})} \\
&= \frac{\int p(x_t|x_{t-1})p(x_{t-1}|Z_{t-1})p(Z_{t-1})dx_{t-1}}{p(Z_{t-1})} \\
&= \int p(x_t|x_{t-1})p(x_{t-1}|Z_{t-1})dx_{t-1}
\end{aligned}
\tag{11}
$$

In the second step, the observation $\mathbf{z}_t$ at time t and the predicted state $\mathbf{x}_t$ can be used to compute the posterior estimate of the state $\mathbf{x}_t$ using

$$
\begin{aligned}
p(x_t|Z_t) &= \frac{p(x_t,Z_t)}{p(Z_t)} \\
&= \frac{p(x_t,z_t,Z_{t-1})}{p(z_t,Z_{t-1})} \\
&= \frac{p(z_t|x_t,Z_{t-1})p(x_t,Z_{t-1})}{p(z_t|Z_{t-1})p(Z_{t-1})} \\
&= \frac{p(z_t|x_t)p(x_t|Z_{t-1})p(Z_{t-1})}{p(z_t|Z_{t-1})p(Z_{t-1})} \\
&= \frac{p(z_t|x_t)p(x_t|Z_{t-1})}{\int p(z_t|x_t)p(x_t|Z_{t-1})dx_t}
\end{aligned}
\tag{12}
$$

Equations 11 and 12 form the optimal Bayesian solution for the recursive propagation of the posterior density. This problem can be solved analytically using the closed-form Wiener-Kalman linear Minimum Mean Square Estimate (MMSE) in Gaussian noise [48, 49]. Non-analytical methods, such as grid-based methods, can be used if the state space is discrete and consists of a finite number of states. For non-linear models, the Extended Kalman Filter (EKF) computes the Jacobian for a Taylor Series expansion of the system and observation models about the current state [50]. Recently, the Unscented Kalman Filter (UKF) has been replacing the EKF in a wide range of applications. The UKF, instead of explicitly computing the Jacobian, computes a set of points that capture the true mean and covariance of the prior. When propagated through the non-linear system, these points capture the posterior mean and covariance [51]. As a result, the UKF estimates the posterior mean and covariance accurately to at least the second-order Taylor Series expansion. The EKF on the other hand achieves only first-order accuracy [52, 53]. More recently,

particle filters which use point mass representations for probability densities and are based on stochastic sampling have been introduced in the visual tracking literature [54, 55]. A primary difference between the UKF and the particle filter is that the former is based on deterministic sampling while the latter is based on stochastic sampling. Particle filters offer an additional advantage of being able to handle arbitrary densities as shown in Figure 15. However, since the particle filter uses non-parametric densities with no functional representations, its computations do not scale well as the dimensionality increases [56].

A variety of particle filters have now been introduced. According to [12], sequential Monte Carlo (SMC) filtering has been called particle filtering [57], bootstrap filtering [54], the condensation algorithm [20], interacting particle approximations [58, 59] and survival of the fittest [60].

In this section, we discussed the Bayesian approach to tracking. This approach can be used for point, region or contour tracking.

3.2 Point tracking

The problem of point tracking in the field of computer vision is similar to the problem of point tracking in the field of radar signal processing. This problem has been extensively studied and is also known as the *data association* problem. In Computer Vision, an additional pre-processing step, *interest-point detection*, is required to extract points of interest from the scene. Point tracking methods can be broadly categorized into statistical methods and deterministic methods.

There are three main statistical methods for point tracking. The *Probability Data Association Filter (PDAF)* provides a computationally efficient method of data association for single targets [13]. It is largely based on the Kalman Filter. However, it has a mechanism of accounting for clutter. The primary difference between the Kalman filter and the PDAF algorithm is in the computation of the innovations process during the update

stage. For multiple targets, this algorithm is generalized by the *Joint Probability Data Association Filter (JPDAF)* [14]. The *Multiple Hypothesis Tracker (MHT)* handles multiple targets in non-linear conditions [15]. MHT requires ever-expanding memory as more and more data is processed. However, computationally practical versions of MHT have been reported [61, 62]. It may be noted that a carefully designed MHT can provide better performance than the PDAF and JPDAF algorithms [63]. Additional details can be found in a survey by Cox [19].

Statistical methods have a number of shortcomings. First, the assumption that points move independently is not always valid. Second, measurements are not always distributed normally around their predicted position. Third, there are a number of parameters to estimate, such as apriori probabilities for false measurements and missed detections. And finally, statistical methods can be computationally demanding [64]. To deal with these shortcomings, a number of researchers have worked on deterministic methods for point tracking.

Most deterministic methods minimize a cost of associating a target to an observation. This correspondence cost is usually a combination of several constraints [2]: (a) proximity, (b) maximum velocity, (c) smooth motion, (d) common motion, and (e) rigidity. A *zero-scan* algorithm uses one frame for correspondence and picks the maximum likelihood hypothesis at every time frame. On the other hand, a *multiple-scan* algorithm uses multiple frames for correspondence [15]. Sethi and Jain [65] use path coherence and motion smoothness for solving the correspondence problem. Unlike many previous approaches, they solve the correspondence problem using multiple frames rather than two. Salari and Sethi [66] extend this work to handle occlusions. Rangarajan and Shah [67] use a greedy non-iterative algorithm with a fixed number of feature points while allowing for temporary occlusion and missing point detections. In [64], a *common-motion* constraint is introduced. According to this constraint, two points lying on the same object should move coherently. Finally, a generic and widely used one-to-one correspondence algorithm is the Hungarian

algorithm [68].

3.3 Region tracking

Several methods for tracking regions have been proposed in the literature. We describe some of the more popular methods in the next few paragraphs.

Template matching is one of the most common methods for tracking regions. This method involves matching the sub-region of an image with a template. A variety of distance measures have been used in template matching. Some commonly used distance measures are SSD (sum of squared differences), SAD (sum of absolute differences) and correlation (CORR). These measures can be computed using the following equations:

$$\begin{aligned} SSD(x,y) &= \sum_{x'} \sum_{y'} [I(x+x', y+y') - t(x', y')]^2 \\ SAD(x,y) &= \sum_{x'} \sum_{y'} |I(x+x', y+y') - t(x', y')| \\ CORR(x,y) &= \sum_{x'} \sum_{y'} I(x+x', y+y') t(x', y') \end{aligned} \tag{13}$$

SAD is widely used in the video coding literature. As a matter of fact, it is used in almost all current video codecs, i.e., MPEG-1, MPEG-2, MPEG-4, H.261, H.263, and H.264. The method of template matching has been applied in many areas of Computer Vision, including tracking. Some examples of the usage of template matching are head tracking [69], motion identification [70, 71], contour matching [72], human detection [73], pedestrian detection [74], finger tracking [75], object recognition [76] and track initialization [70, 77]. The advantages of template matching are simple implementation and robustness to short-term, gradual changes in appearance. The disadvantages of template matching are failure under occlusions, and the need for updating over time. A solution to the template updating problem is presented in [70].

A number of researchers have used density based methods to track regions. A commonly used method of density tracking is the *mean-shift* algorithm. In this algorithm, the

gradient vector of the density is computed. Repeated iterations lead to a local mode of the density [78].

Finally, *optical flow* is a method of computing the motion between two successive frames. Since motion computation is an ill-posed problem, two constraints are commonly used to compute a solution: (a) brightness constraint, i.e. $I(x, y, t) = I(x + \delta x, y + \delta y, t + \delta t)$ and, (b) smooth velocity constraint, i.e. $\nabla^2 u + \nabla^2 v$, where $I(x, y, t)$ is an image pixel in image I at location (x, y) at time t, $u = dx/dt$ is the velocity in the x-direction, and $v = dy/dt$ is the velocity in the y-direction. An iterative scheme to compute optical flow is given by Horn and Schunk [79]. An alternate method is given by Lucas and Kanade [80]. A comparison of optical flow techniques can be found in [81]. Motion information is a useful feature in tracking. However, in many tracking scenarios, optical flow assumptions of brightness constancy do not hold. This is particularly true when multiple targets are being tracked and occlusions are common. Nevertheless, some attempts at robust tracking have been made in these difficult situations [82].

3.4 Contour tracking

The third and final tracking method we discuss is contour tracking. A method of representing a closed contour using ellipses as basis functions is the Elliptical Fourier decomposition [83] given by

$$
\begin{aligned}
a_0 &= \frac{1}{2\pi}\int_0^{2\pi} x(t)dt, \\
c_0 &= \frac{1}{2\pi}\int_0^{2\pi} y(t)dt, \\
a_k &= \frac{1}{\pi}\int_0^{2\pi} x(t)cos(kt)dt, \quad b_k = \frac{1}{\pi}\int_0^{2\pi} x(t)sin(kt)dt \\
c_k &= \frac{1}{\pi}\int_0^{2\pi} y(t)cos(kt)dt, \quad d_k = \frac{1}{\pi}\int_0^{2\pi} y(t)sin(kt)dt \\
\begin{bmatrix} x(t) \\ y(t) \end{bmatrix} &= \begin{bmatrix} a_0 \\ c_0 \end{bmatrix} + \sum_{k=1}^{K} \begin{bmatrix} a_k & b_k \\ c_k & d_k \end{bmatrix} \begin{bmatrix} cos(kt) \\ sin(kt) \end{bmatrix}
\end{aligned} \tag{14}
$$

However, due to the difficulty in incorporating prior information in this representation, as well as the non-intuitiveness of the computed Fourier coefficients, this method has not been widely used for contour tracking [84]. A widely used method is the *snakes* method [85] in which a parameterized curve $\mathbf{v}(s) = (x(s), y(s))$, such as a B-spline curve, is allowed to evolve while minimizing an energy functional,

$$
\begin{aligned}
E_{snake} &= \int_0^1 E_{snake}(\mathbf{v}(s)) \\
&= \int_0^1 E_{int}(\mathbf{v}(s)) + E_{ext}(\mathbf{v}(s))ds
\end{aligned} \tag{15}
$$

where E_{int} is the internal energy of the spline, and can be written as a sum of two terms: (a) membrane energy ($\frac{\alpha}{2}(x_s^2 + y_s^2)$), and (b) thin-plate energy ($\frac{\beta}{2}(x_{ss}^2 + y_{ss}^2)$). E_{ext} is the external energy and is derived from the image, for instance by computing the gradient evaluated along the contour. This additional term makes it possible to incorporate prior information about the image. If the gradient is used, then Equation 15 can be written as

$$
E = \int_0^1 \left[\frac{\alpha}{2}(x_s^2 + y_s^2) + \frac{\beta}{2}(x_{ss}^2 + y_{ss}^2) - \left[\nabla(x(s), y(s))\right]\right] dt \tag{16}
$$

Snakes have been used in a variety of applications, including walker tracking [86], head tracking [87], and vehicle and hand tracking [88]. In order to model more complex shapes,

level-sets were introduced in [18]. In this formulation, a contour is represented as the zero crossings in a level-set grid. The evolution of the contour is governed by changing the grid values [2].

A further extension to the snakes model is the *active-appearance model*, or "smart" snake model, proposed by Cootes et al. [89]. In this approach, structural constraints derived from a training set are imposed on the model. This is done to guide the shape evolution. As a result, this method captures the natural variability within a class of shapes. The advantage over snakes is that shape deformation is always consistent with the training set. This method has also been applied to tracking, for instance to human tracking [90, 91].

3.5 Example trackers

The tracking techniques mentioned in the last three sections, i.e. point tracking, region tracking and contour tracking, have been applied to a variety of applications. In particular, a lot of attention has been given to human tracking. In the following few paragraphs, we summarize the major research in this important area.

- KidsRoom [92]: This system tracks little children in a reasonably realistic environment. A background model is used to generate blobs. Four features are used for tracking: average normalized color, distance, velocity and size.

- Pfinder [93]. This work popularized background subtraction [2]. The advantage of background subtraction is that it can lead to a blob based representation. Such a representation reduces the degrees of freedom in going from individual pixels to blobs. The Pfinder system tracks a single user. The features used for tracking include skin color and 2D shape contour for head, hands and feet. For each pixel, a likelihood is computed for each of the blob models. The class membership likelihoods are resolved using spatial and connectivity constraints. A major drawback of this system is the process of initialization in which a user is required to carry out specific actions.

- CMU [70]. This is a classification and tracking system. This system is based on the fact that properties of template matching and temporal differencing (DT) are complementary. When the target is stationary, template matching is at its most robust, while DT does not do well. If the target is moving, template matching does not do well while DT does. This system classifies and tracks humans and cars based on size and *dispersedness* ($\frac{perimeter^2}{area}$). MLE is used for classification. The templates are updated using an IIR filter.

- W4 [94]: This system detects foreground objects using a background, bimodal-Gaussian, intensity distribution. No use of color is made. In comparison, Pfinder uses color information. Another difference is that unlike Pfinder, W4 does not assume that there is only one person in the scene. The features used are shape and appearance models. Besides tracking humans, this system can recognize simple events such as carrying, leaving or exchanging bags.

- Bramble [95]. This system uses a known camera model and ground plane. It can therefore track humans in 3D. The state estimation is done using particle filters.

- Zhao and Nevatia [56]. This tracker simultaneously tracks up to 13 people in a crowd. The researchers use a color-histogram based mean-shift tracker for appearance model correspondence. The human body is modeled using a 3-ellipsoid, one for the head, one for the torso, and one for the legs. The prior has spatial and temporal components. The spatial prior penalizes unnecessary overlapping of targets and blobs with small sizes. The temporal prior encourages smoothness and trajectory connectivity. The likelihood is computed using background exclusion and correspondences. The MAP estimate is computed using Markov Chain Monte Carlo (MCMC) sampling. State estimation for velocity is done using the Kalman filter.

- Brostow and Cipolla [96] In this tracker, an unsupervised Bayesian clustering method

is used to detect individuals moving in crowded scenarios. Interestingly, the only feature used is motion. No training data is used, nor is there any appearance model. The idea is to probabilistically cluster regions moving in unison. Motion initialization is done using optical flow. Subsequently, normalized cross correlation is used to match corners. Regions with similar motion are clustered to form targets.

3.6 Subspace based tracking

Subspace based tracking is a form of region tracking. It is discussed here separately from region tracking since we use this approach in this work. Therefore, this method is discussed in more detail.

One of the main factors limiting visual tracking is the lack of suitable appearance models [97]. In subspace based tracking, the basic assumption is that the evolving target appearance can be modeled using a lower dimensional subspace computed using PCA, or a few code-vectors computed using a VQ based method. This approach has the following advantages:

- **Compact representation**. A subspace representation using say PCA or RVQ allows storage of a few basis eigenvectors or stage code-vectors to capture variations in the target appearance.

- **Object recognition**. This method facilitates object recognition since an appearance model is built for each target.

- **Continuous model update**. As mentioned earlier, changes in target appearance are a big challenge in target tracking. Online model updating allows the target appearance to be built dynamically.

- **Less offline training data required**. Since the models are built online, less training examples are typically required.

- **No optimization**. No complex optimization is required.

A disadvantage of this approach is that the tracker is prone to drift if the online appearance model is updated incorrectly. In [2], subspace based tracking is mentioned under the category of view-based approaches. View-based approaches can be interpreted to mean one of three approaches:

1. Multi-camera approach. As the name suggests, several cameras are used to generate simultaneous views of an object from different angles [98].

2. VBR approach. In this approach, a limited number of views of an object are sampled as it is rotated about the x, y and z axes of rotation during a training phase. This approach forms the basis of view-based recognition (VBR) and allows replacing the matching of a single 3D model with matching a large number of 2D models [7, 99].

3. Online appearance approach. In this approach, several views of an object are captured as it is tracked online and used to model the dynamic appearance of the object [1].

In this work, we do not focus on the multi-camera approach. We start our discussion with the VBR approach and transition to the online appearance approach for two reasons: (a) it historically predates the online appearance approach, and (b) it naturally leads to the online appearance approach, which is the approach that we take.

In order to classify or recognize complex articulated objects, a large range of appearances are required. One approach has been to use interpolation of appearance from a small number of views. [6] makes the assumption that all possible views of an object after 3D transformations such as rotation, translation and scaling can be expressed as the linear combination of other views of the same object. Therefore, object matching is done by finding the distance between the linear subspace (or low dimensional manifold) defined by previous views and an observed object, rather than measuring the distance between the object and each of the stored views. Figure 16 shows an example of generating different car views using linear combinations of existing views.

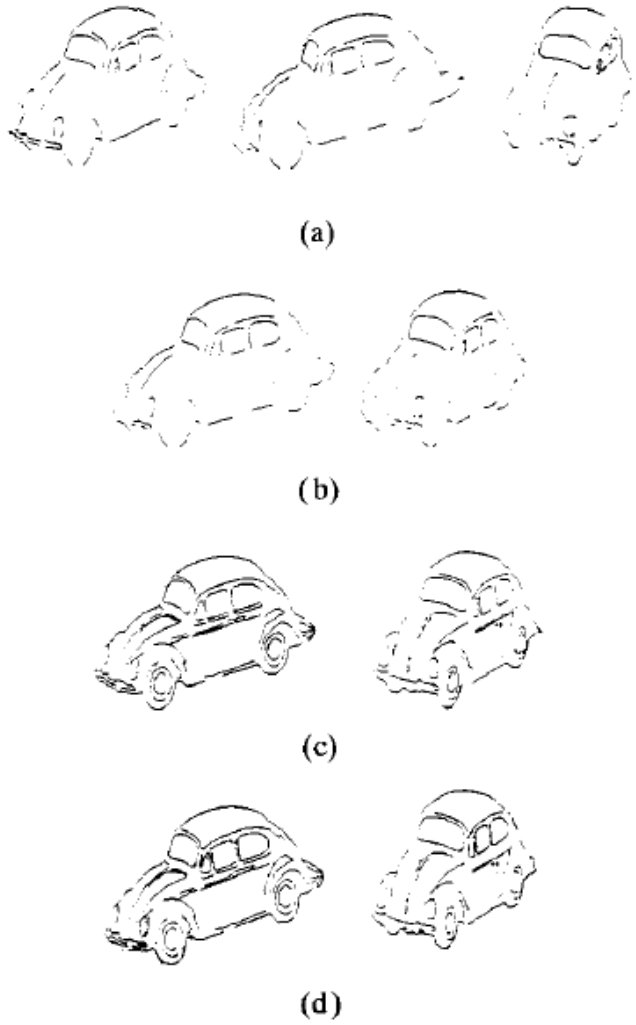


Figure 16. (a) Original images, (b) Synthetic images created from linear combinations of original images, (c) More original images at approximately same orientation as the synthetic images, (d) Synthetic images superimposed on original images [6]

.

A step forward in the justification of view-based representations is presented in [7], where it is shown that 300 views need to be stored for each view-based model to achieve an error rate smaller than that of optimal 3D matching algorithms. An example of such a matching is shown in Figure 17.

A fundamental question here is how to learn an appropriate set of view models. As mentioned earlier, in traditional tracking approaches, such as normalized correlation or template matching, there is a limitation that the image motion must be simple, such as

Figure 17. Airplane recognized by view-based recognition system [7]

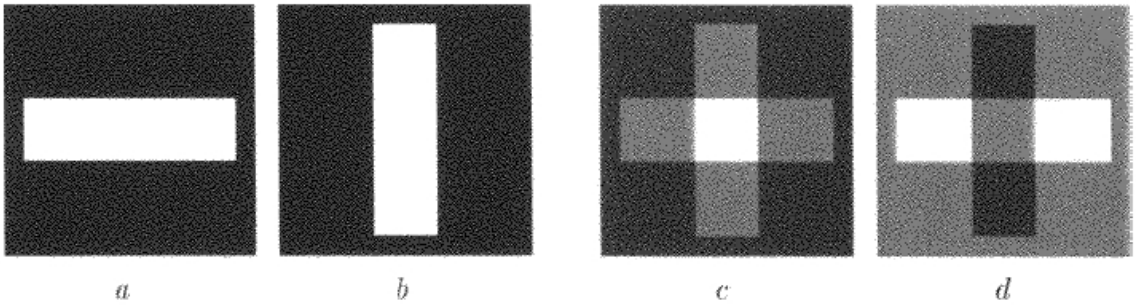


Figure 18. (a, b) Training images (c, d) Eigenspace basis images [8]

translation and the viewpoint must be fixed or changing slowly. In [99], this challenge is tackled by using sets of view models, rather than simple templates. In this approach, a data-driven method of using normalized correlation scores to automatically construct a set of view models is developed. One initial model is specified by the user using a cursor. The target object is tracked using normalized correlation. The search function correlation scores are saved so that when they fall below a certain threshold, a new model can be added to the search set using the image at the offset with the best current score. Over time, a family of view models that sample the aspect space are accumulated. Two thresholds are maintained, one for deciding if track has been lost, and the other sets the level at which a new model should be added. This method is then used to recognize human gestures.

A multi-view method that deals with illumination changes, to which SSD (sum of

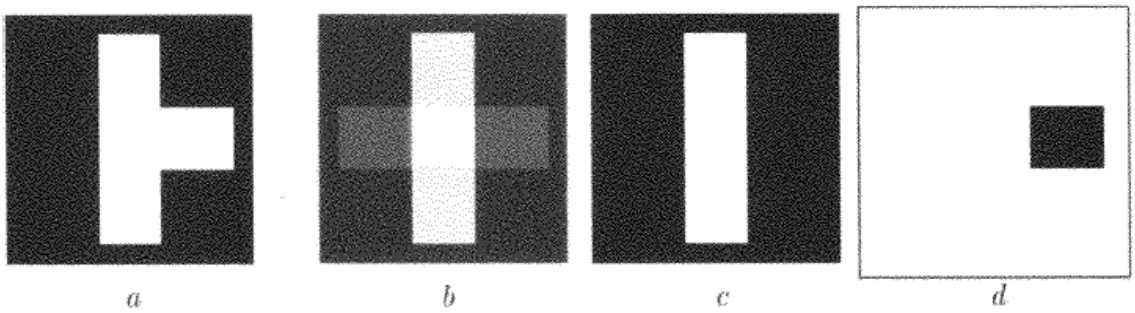


Figure 19. (a) Test image (b) Least squares reconstruction (c) Robust reconstruction [8]

squared differences) or correlation based tracking is sensitive to, is presented in [100]. A set of 5 basis images is created offline for a single face under a single view but different illumination conditions. Images with maximum singular values in the SVD decomposition are retained for the basis. These basis images are then used to approximate the object under any illumination condition. This work also accounts for geometric changes in the face through affine warping.

The transition of VBR to view-based tracking is first made in the seminal work of *eigentracking* presented in [8]. This is an adoption of initial work in the area of PCA based methods to efficiently represent several views [89, 101]. Before this work, eigenspace representations had focused on the problem of object recognition and had only peripherally addressed the problem of object tracking over time. Additionally, it was assumed that the object of interest could be located in the image, segmented and transformed into canonical form for matching with the eigenspace. However, this is not always possible and eigenspace reconstruction methods are not invariant to image transformations such as translation, scaling and rotation. Two primary observations are made in this work that have formed the basis of current subspace tracking methods:

1. Robust estimation. PCA reconstruction relies on a least squares fit between an image and the eigenspace. This can lead to poor results in the presence of structured noise. This work reformulates the eigenspace matching problem as one of robust estimation.

2. Affine transformation. Instead of storing all views of the object to be tracked or learn

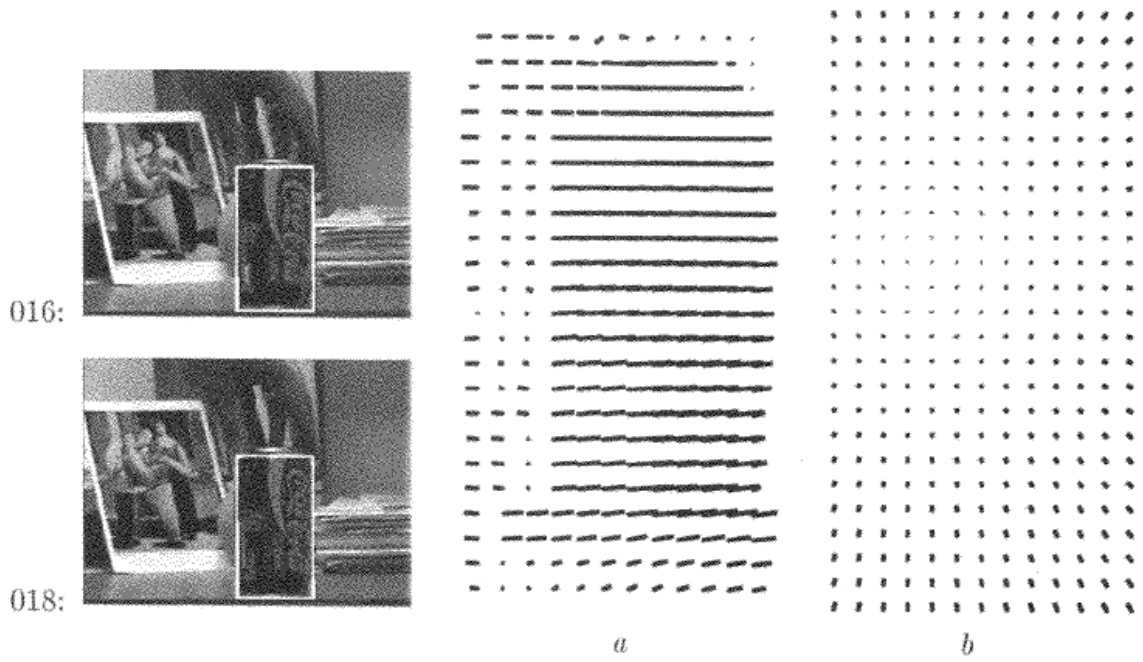


Figure 20. Brightness versus subspace constancy. "Motion" between frames 16 and 18 (computed within the white boxed region) (a) dense optical flow for the soda can computed using the brightness constancy assumption (b) "Flow" computed using the subspace constancy assumption for the same frames [8]

interpolating surfaces in the eigenspace, an affine transformation is allowed between the input image and the subspace.

A fundamental issue of whether to create one eigenspace for all classes or one eigenspace per class is addressed in [10]. The classes correspond to M human head orientations with N examples in every class. In a *parametric* eigenspace, one eigenspace is created for all NM images. On the other hand, in a *view-based* eigenspace, one eigenspace is created for each of M head orientations, each with N users per eigenspace. Since multiple views of a face form a connected non-convex region [102], the analogy of using a parametric versus a view-based eigenspace approach is that of modeling a complex distribution by a single cluster model or the union of several component clusters respectively. It is expected that the latter approach will give better image reconstruction results as seen in Figure 21.

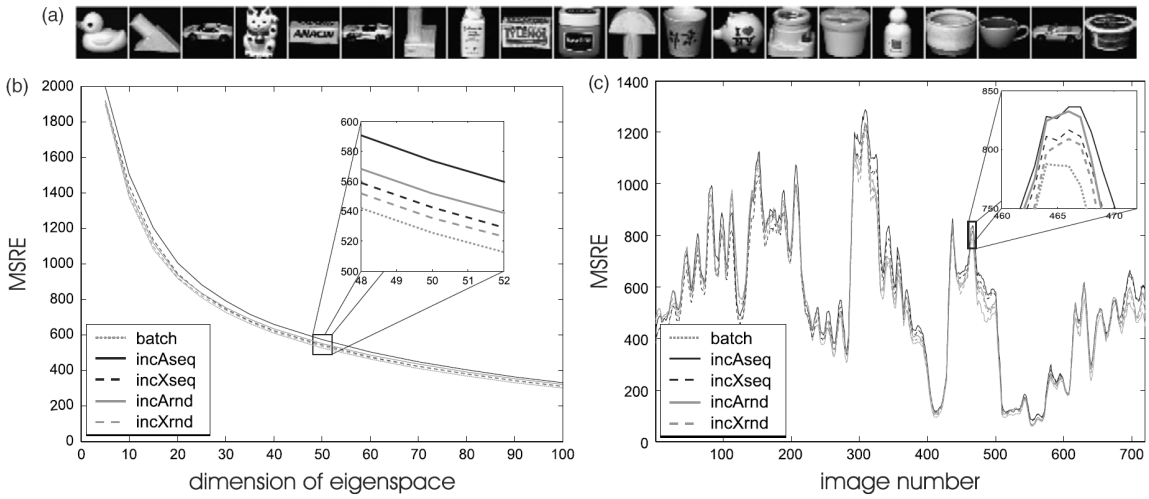
Recently, a tracker based on online updating of a PCA eigenspace was presented in [1].

(a) Training image. (b) Training image reconstruction, parametric eigenspace (c) Training image reconstruction, view-based eigenspace

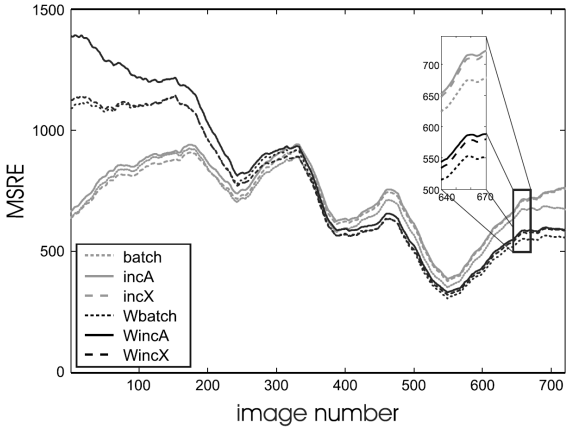
(d) Test image. (e) Test image reconstruction, parametric eigenspace (f) Test image reconstruction, view-based eigenspace

Figure 21. Comparing parametric eigenspace with view-based eigenspace.

In this work, the authors use incremental PCA and a particle filter to build an online incremental basis. It is assumed that observations are generated from this eigenspace. In [97], phase is chosen as the basis of the appearance model since it provides some amplitude and illumination independence. An optimal image warp is computed from the stable properties of image appearance. A $\mathcal{WSL}$ tracker is created, where $\mathcal{S}$ refers to the stable component, $\mathcal{L}$ refers to the "lost" component, and $\mathcal{W}$ refers to the wandering component. The $\mathcal{S}$ component captures the behavior of the stable and slowly varying image observations when and where they occur. The $\mathcal{L}$ component accounts for data outliers which arise as a result of tracking failures due to tracking, occlusion, or noise. The $\mathcal{W}$ component allows for adaptation to short term image appearance changes. A mixture model is used to model these 3 components. The image feature used to generate the mixture model is the phase. The model is learned online using the EM algorithm. This method can handle variations in



(a) In the image on the right, dimension is fixed to 50



(b) Comparison of weighted vs non-weighted batch and incremental PCA algorithms

Figure 22. Comparison of batch PCA with various incremental PCA update algorithms [9]

pose, illumination and expression. However, since pixels are treated independently, within the target region, a notion of an object being tracked does not exist. This can result in modeling the background as well.

In [9], the researchers try to find the error associated with using an incremental PCA update algorithm versus using the optimal batch PCA algorithm. They build eigenspaces of various dimensions from 720 images of 20 objects taken from the Columbia University Image Library (COIL-100) [103]. Figure 22 shows the mean squared reconstruction errors

(MSRE) for the batch and various incremental methods. In all experiments, squared reconstruction error degradation is less than 10%. Moreover, the sequential order in which the images are presented influences results. It is pointed out that to obtain good results, the initial images should be heterogeneous encompassing different objects and views. This results in an evolving eigenspace that is rich and comprehensive enough in the beginning and is not specialized for representing a specific object only. In this work, it is also shown that the reconstruction errors of the various incremental weighted PCA methods (*WincA, WincX*) do not differ significantly from the results of the batch weighted method (*Wbatch*). This method is used by [104] for object tracking in a manner quite similar to the approach used in [1]. They sample a collection of image patches and likelihood of each image patch is generated by reconstruction. Comparison is made between PCA subspace tracking with and without weighting prior observations. They show that temporal weighting the data results in less background clutter penetrating the target of interest and therefore leads to better occlusion handling in tracking. Another incremental PCA update algorithm is developed by [105]. They also propose an incremental algorithm robust PCA in addition to standard PCA.

Having discussed RVQ and tracking methods, we now turn to the usage of RVQ in tracking.

CHAPTER 4

RVQ TRACKING

4.1 Introduction

In this chapter, we combine information on target representations presented in Chapter 1, theoretical knowledge of RVQ presented in Chaper 2 and an overview of tracking methods presented in Chapter 3 into a visual tracking framework using RVQ and compare it with visual tracking using PCA and TSVQ.

This work is significant since PCA is commonly used in the Pattern Recognition, Machine Learning and Computer Vision communities. On the other hand, TSVQ is commonly used in the Signal Processing and data compression communities. RVQ with more than two stages has not received much attention due to the difficulty in producing stable designs. In this work, we use a multi-stage RVQ designed by Barnes [39] and integrate it into a learning based tracker. This is then compared with PCA based and TSVQ based trackers. The result is robust tracking for all 3 methods, but with RVQ performing the best according to certain defined criteria to be described later in the chapter. Moreover, an advantage of our approach is a learning-based tracking framework that builds the target model while it tracks, thus avoiding the costly step of building target models prior to tracking [106].

The approach we take in this work builds on work presented by Ross et. al. in 2008 [1]. We have used part of their software with their permission [107]. In this spirit, we make our software available for download to the community at `https://github.com/SalmanAslamPhD/PhD`.

4.1.1 Challenges

Visual tracking is the task of estimating a target's state over time. In many cases, the "target state" can be defined to represent target position, a bounding box, or the target contour. This is a challenging problem due to the following reasons:

1. Appearance and contour changes: A target of interest can undergo change in appearance and contour. This can be due to the following reasons:

   (a) Pose change: The target can rotate and present a different view to the camera.

   (b) Warping: The target can undergo warps, such as expression changes for humans.

   (c) Self occlusion: The target can be occluded or unoccluded by itself or its surroundings.

   (d) Blur: Motion blur can severely distort a target's appearance.

   (e) Structured noise: The target can change appearance in an orderly manner, for instance, a target of interest can put on or remove glasses or a hat.

   (f) Random noise: This can be a result of atmospheric effects in the optical channel, sensor noise, electronics noise and EMI (electromagnetic interference). On the software side, it can be caused by compression artefacts.

   (g) Non-symmetic BRDF: The light reflected off an object as a function of direction is modeled by the bidirectional radiation transfer function (BRDF)[1]. Since this function may not be symmetric in all directions, the amount of light reflecting off the object may be different in different directions. In such cases, multiple cameras viewing the same point will receive different intensity levels.

2. Lighting change: Lighting changes can be caused by turning on or off lights in indoor environments, or moving into or out of shades in outdoor environments.

3. Sudden motion (target or camera): Besides motion blur, sudden motion by the target or camera can cause the target to exit the window in which the tracker looks for the

[1]BRDF is given by $\rho(\theta_o, \phi_o, \theta_i, \phi_i) = \frac{L_o(x,\theta_o,\phi_o)}{L_i(x,\theta_i,\phi_i)\cos\theta_i d\omega}$, where the angles (θ_o, ϕ_o) define the outgoing light direction and angles (θ_i, ϕ_i) define the incoming light direction. A surface illuminated by radiance $L_i(x, \theta_i, \phi_i)$ coming in from a differential region of solid angle $d\omega$ at angles θ_i, ϕ_i receives irradiance $L_i(x, \theta_i, \phi_i)\cos\theta_i d\omega$. Irradiance is measured in W/m^2, while the solid angle $d\omega$ is measured in steridians, sr. The unit of BRDF is therefore sr^{-1} [108].

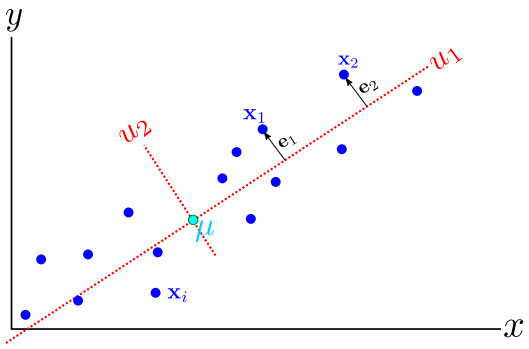


Figure 23. In $\mathbb{R}^2$, a reduced eigenspace means that eigenvector u_2 is discarded. Vectors $\mathbf{x}_1$ and $\mathbf{x}_2$ have the same projection error on eigenvector u_1 even though $\mathbf{x}_1$ is closer to the mean μ of the training data $\mathbf{x}_i$.

target leading to incorrect track assignment.

For a tracker that tries to learn the appearance and/or contour model of the target, inclusion of background pixels is an added problem that can cause track drift. If none of the problems mentioned above are present, a simple template matching stategy would suffice for robust tracking. This has complexity $O(Knm)$, where K is the number of templates, n is the number of pixels in the target and m is the number of locations at which the target is searched. For most practical situations, where K is sufficiently small, this does not represent significant computational load and can be done in real time even while tracking several targets. However, in the presence of several forms of noise, which is generally the case in tracking applications, more sophisticated methods are required. In this chapter, we focus on methods involving compact representations of the target appearance model, i.e., PCA, TSVQ and RVQ.

4.1.2 Brief history

In this work, we try to address single-target visual-tracking under several of the challenges mentioned above while trying to learn the appearance model of the target. Seminal work here can be traced back to 1996 when Black and Jepson experimented with tracking using an eigenspace representation of the target appearance model [8]. The next notable work is by Moghaddam and Pentland, 1997 [10] in which they try to address a fundamental

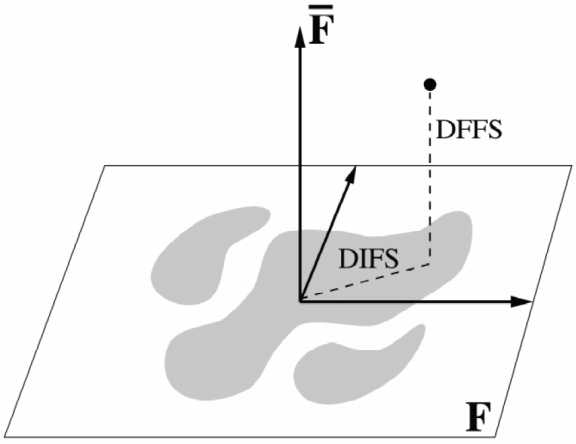


Figure 24. Graphical illustration of DFFS (distance-from-feature-space) and DIFS (distance-in-feature-space). The feature space is F while the subspace orthogonal to the feature space is $\bar{\mathbf{F}}$. DFFS is the signal residual error and DIFS is the F-space likelihood [10].

limitation of PCA. This fundamental limitation is illustrated by the following example. In PCA, 2 vectors, $\mathbf{x}_1$ and $\mathbf{x}_2$ can have the same distance to a reduced eigenspace, i.e., projection error $\mathbf{e}_1$ and $\mathbf{e}_2$ respectively, even if they have different distance to the mean $\boldsymbol{\mu}$ of the data that was used to create the eigenspace. This is shown for a trivial case in $\mathbb{R}^2$ in Figure 23 where $\mathbf{e}_1 = \mathbf{e}_2 = \mathbf{e}$ even though $\mathbf{x}_1$ is closer than $\mathbf{x}_2$ to the mean $\boldsymbol{\mu}$. They formulate the problem using DIFS (distance in feature space) and DFFS (distance from feature space) so that both projection error and within-subspace distance to the mean of the data are used while trying to determine how well the subspace explains a new data-point. The next breakthrough came with the work of Bishop and Tipping in 1999 [109], where they show that a probabilistic variation of PCA, probabilistic PCA (PPCA), allows PCA to be used as a generative model. The advantage in tracking is that this methodology allows an assignment of probabilities to new data-points and therefore allows relative weighting of track candidates. Ideas from these three works were combined into a tracking framework by Ross et. al. in 2008 [1]. Moreover, they used incremental SVD to make their tracker run in real time.

Here, we extend the work in Ross et. al. [1] using RVQ in a similar tracking framework and comparing it with PCA and TSVQ based tracking. We also introduce four methods for

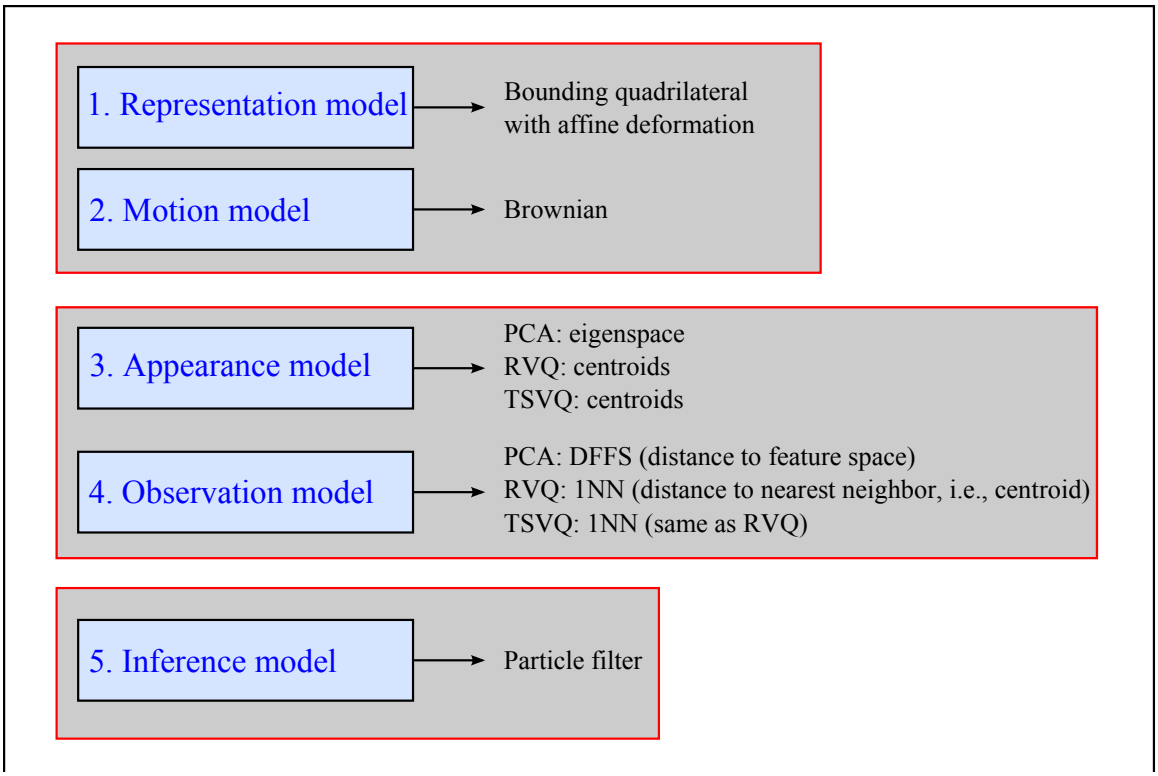


Figure 25. Tracking framework, overview.

relative weighting of track candidates for RVQ. The result is a generative framework for RVQ that leads to robust tracking. Whereas RVQ was first introduced by Juang and Gray in 1982 [27], and subsequently extended by the work of Barnes [39, 35, 40, 41, 38, 42, 43, 37, 32, 44, 28, 5], this algorithm has received little attention outside the signal processing and data compression communities. In this work, our goal is to introduce RVQ in the computer vision and machine learning fields where a much simpler cluster-means (K-means) based classifier has been widely used [110]. We present RVQ in the context of an important and challenging problem, that of visual target tracking.

4.1.3 Overview of approach used

Our goal is to produce estimates at every time frame of the target state, i.e. 6 affine parameters that define a target bounding quad. In order to accomplish this, our tracking framework is based on five components as shown in Figure 25 . Each of these components is described in detail in later sections. Here, we present an overview of each:

1. Representation model. The goal of the representation model is to provide a means of specifying a target. Several target representation methods are described in Chapter 1. We use the bounding quadrilateral method. This quad encloses the pixels of a target of interest. It is also allowed to warp affinely from frame to frame to minimize inclusion of background pixels as the target changes shape, size and orientation.

2. Motion model. The goal of the motion model is to specify the motion that the target is expected to follow. In order to keep our work general, we do not assume any deterministic target motion model. The target is expected to move according to a Wiener process, i.e., brownian motion. This allows for robust tracking under arbitrary target and camera motion.

3. Appearance model. The goal of the appearance model is to provide a compact representation of the target's pixel intensities. In this work, we use a learned eigenspace for PCA, a trained σ-tree codebook for RVQ and a binary balanced-tree codebook for TSVQ.

4. Observation model. The goal of the observation model is to (a) generate observations based on the motion and representation model outputs, and (b) generate a likelihood score for each observation using the appearance model. For PCA, the likelihood of a target observation is assumed proportional to the DFFS (distance to feature space). For RVQ, the likelihood of a target observation is assumed to be proportional to the Euclidean distance to a direct-sum code-vector reached through sequential search. We use two methods, *maxP* and *RofE*, to compute the full-stage direct-sum code-vectors, and two methods, *nulE* and *monR* to compute the partial-stage direct-sum code-vectors. These methods are explained later in this chapter. For TSVQ, the likelihood of a target observation is assumed to be proportional to the Euclidean distance to the closest terminal code-vector.

5. Inference model. The goal of the inference model is to: (a) weight the likelihoods of

various observations and make a decision on which observation should be picked as an estimate for the target position and appearance, and (b) keep a temporal record of which observations were not picked in the previous frames as best estimates but may still potentially be considered in future frames. In tracking, the correspondence of observations in the current frame to existing targets in the previous frame is generally an ill-posed problem [111]. We use the particle filter to deal with this problem by propagating multiple hypotheses from frame to frame [20]. The computational complexity of this method does not grow with frames as opposed to the multi-hypothesis tracker (MHT) [19].

These models work together to produce state estimates at every time frame. The motion model and the representation model work together to generate 600 affine parameter sets as candidates for the target state. The observation model takes these affine parameter sets and extracts observations, i.e., candidate window-chips, also called *snippets* [5] from the image. It then uses the appearance model to generate a likelihood score for each snippet. Finally, the inference model picks the snippet with the highest score and goes through a resampling step so that snippets with low likelihood scores are eliminated. The affine parameters of the resampled snippets are then given to the motion model in the next frame and the process continues.

4.1.4 Overview of temporal process

In the previous section, we described the five major components of our visual tracking framework and their mutual interaction. In this section, we describe the temporal evolution of the tracking process. Refer to Figure 26 for a graphical overview. The temporal process is based on 3 distinct phases:

1. Manual initialization. The target is identified in the first frame by manually drawing a bounding box around it and identifying certain feature points on it.[2] For instance,

[2] In future work, this manual process could be replaced with an RVQ detection for features such as eyes

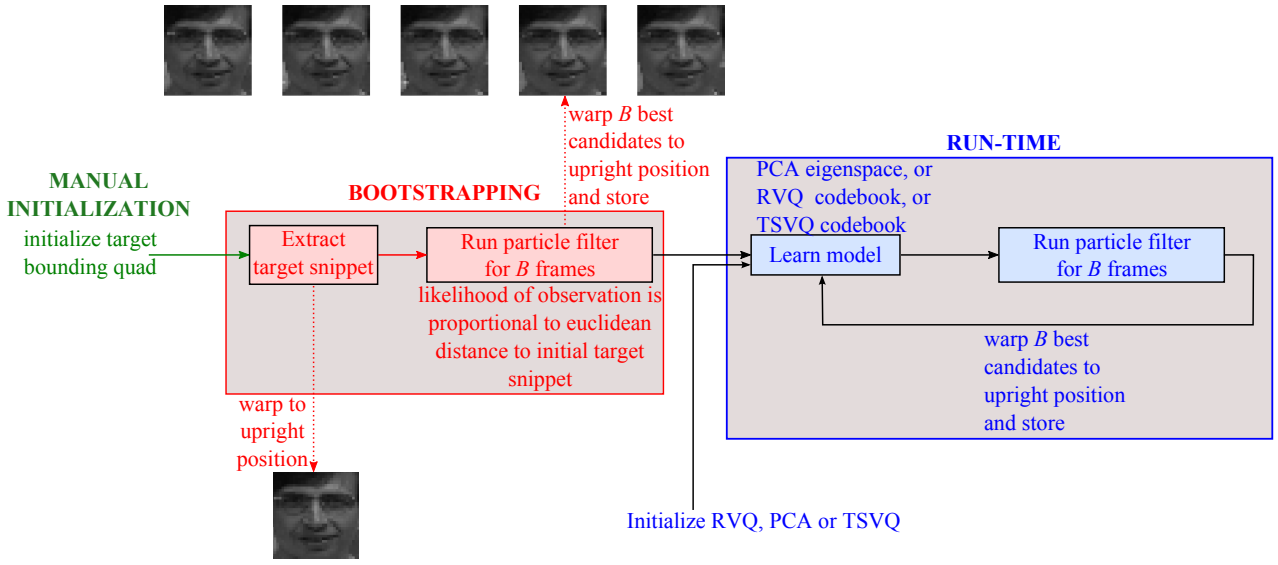


Figure 26. Temporal overview.

for the Dudek sequence in which a face is tracked, feature points include the outer edges of the eye and points on the lips. For the car11 sequence in which a car is tracked from the read, feature points include the tail lights.

2. Bootstrapping. A particle filter is run for B frames. The likelihood of observations is computed using the Euclidean distance from the manually segmented target in the first frame. The B maximum aposteriori (MAP) snippets, one from each of the B frames are stored.

3. Run-time. During this step, the learning process and the particle run alternately. For PCA, the learning process includes updating its eigenbasis with the MAP estimates of the particle filter. For RVQ and TSVQ, the learning process includes updating their codebooks.

Our tracking framework based on the 5 models mentioned in the previous section working through the temporal process explained in this section enable us to handle the following tracking challenges:

- Target appearance related: Target pose changes, lighting changes, structured noise

Table 1. 2D transformations.

Transformation	DoF	Matrix	Distortion
Projective	8	$\begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}$	any arbitrary quadrilateral as long as no three points are collinear
Affine	6	$\begin{bmatrix} a & b & t_x \\ c & d & t_y \\ 0 & 0 & 1 \end{bmatrix}$	rotation and non-isotropic scaling
Similarity	5	$\begin{bmatrix} sr_{11} & sr_{12} & t_x \\ sr_{21} & sr_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix}$	scaling and rigid motion
Euclidean	4	$\begin{bmatrix} r_{11} & r_{12} & t_x \\ r_{21} & r_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix}$	rigid motion (rotation, translation)

and temporary occlusions.

- Target representation related: Target scale and orientation changes.
- Target motion related: Arbitrary camera and target motion.

We now discuss each of the 5 models of our tracking framework (see Figure 25).

4.2 Model 1: Representation model

In this section, we discuss the representation model. See also Figure 2 in the introductory chapter that shows different target representations. In many situations, it is necessary to track a visual target that is undergoing deformations. Several targets of interest fall in this category, particularly non-rigid targets such as humans. Even rigid objects can undergo deformation in a matter of seconds as shown in Figure 27.

(a) Frame 770. (b) Frame 1770.

Figure 27. Over time, even rigid objects can undergo deformations such as the car in these images from the PETS2001 dataset.

In such cases, using a rigid rectangular bounding box to represent the target will inevitably lead to inclusion of background pixels in the matching process. This can easily lead to tracker drift, particularly if the tracker is also trying to learn the appearance model of the target.

We now show how to use affine warping of the rectangular bounding box so that it more closely captures the outline of the target of interest. This minimizes inclusion of background pixels in the matching process and leads to more robust tracking.

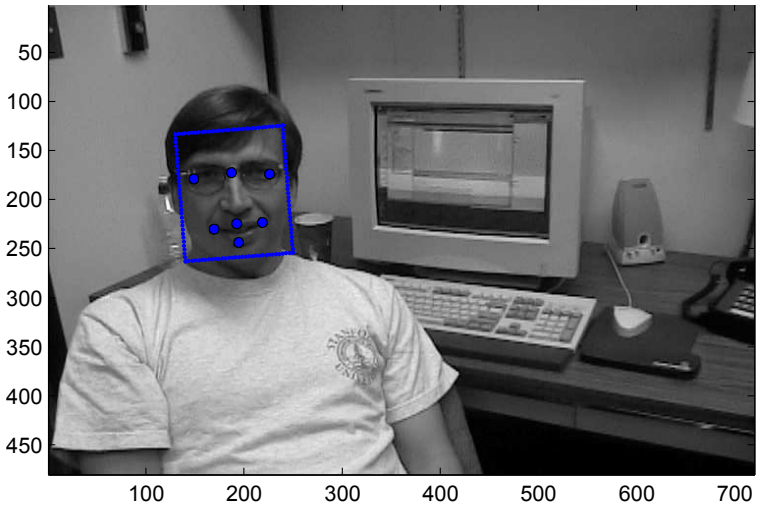
Table 1 shows different kinds of 2D linear transformations. Every transformation generalizes the transformation below it in the table. In this report, we are interested in the 2D affine transform since it is flexible enough to account for most distortions in real images.

The affine transform[3] is given by,

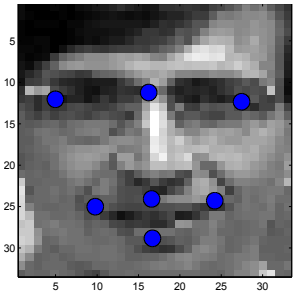
[3]The notation adopted by some for the affine transform is,

$$\begin{aligned} X &= ax + by + e \\ Y &= cx + dy + f \end{aligned} \tag{17}$$

where the input coordinate (x,y) has been transformed through 6 affine parameters, a, b, c, d, e, f to the output coordinate (X, Y). Instead of e and f, we will be using t_x and t_y respectively.



(a) Affine parameters $(\theta, \lambda_1, \lambda_2, \phi, x, y)$ corresponding to the bounding box and a few feature points are manually selected.



(b) Reference position of feature points.

Figure 28. Manual target initialization in the first frame. The manually selected target and manually selected feature points (top image) are warped to an upright reference position using $(\theta, \lambda_1, \lambda_2, \phi, x, y)$. The position of these feature points (lower figure) is kept as reference throughout the tracking process.

$$
\begin{aligned}
\begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} &= \begin{bmatrix} a & b & t_x \\ c & d & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \\
&= \begin{bmatrix} \mathbf{A} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix} \mathbf{x} \\
&= \mathbf{A}\mathbf{x} + \mathbf{t} \\
&= \mathbf{H}_A \mathbf{x}
\end{aligned}
\tag{18}
$$

where t_x and t_y are translations in the x and y directions respectively and $\mathbf{H}_A$ is the

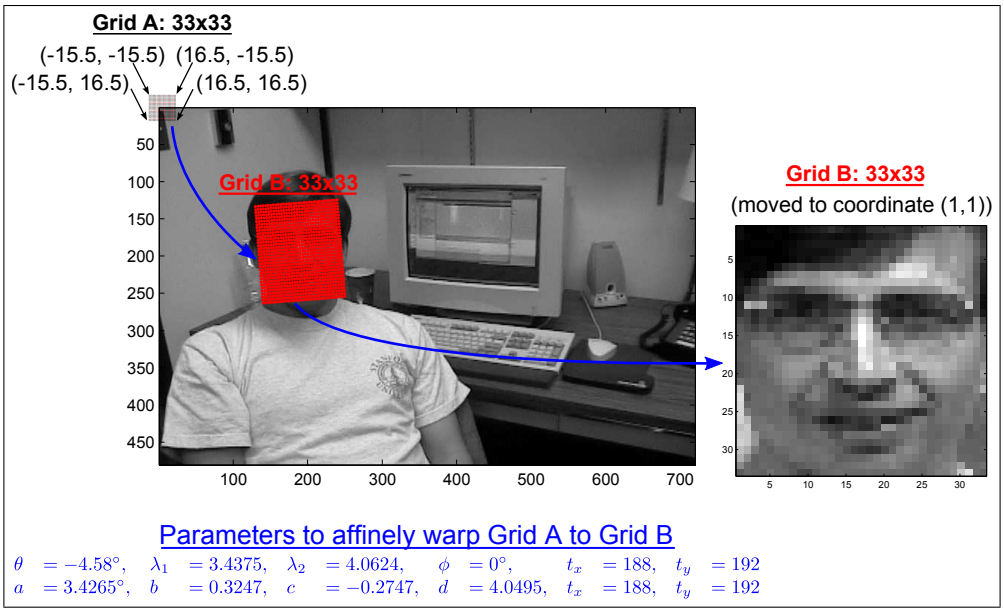


Figure 29. Run-time processing. A zero-centered grid is warped using a given set of affine parameters to cover the object of interest. Pixel intensities at the warped grid points are computed using bilinear interpolation.

affine transformation matrix. The matrix $\mathbf{A}$ above can always be decomposed using the SVD decomposition as the product of orthonormal matrix $\mathbf{U}$ containing the eigenvectors of $\mathbf{A}\mathbf{A}^T$, orthonormal matrix $\mathbf{V}$ containing the eigenvectors $\mathbf{A}^T\mathbf{A}$ and a diagonal matrix $\mathbf{S}$ containing the eigenvalues of $\mathbf{A}$ [112]:

$$
\begin{aligned}
\mathbf{A} &= \begin{bmatrix} a & b \\ c & d \end{bmatrix} \\
&= \mathbf{U}\mathbf{S}\mathbf{V}^t \\
&= (\mathbf{U}\mathbf{V}^t)\mathbf{V}\mathbf{S}\mathbf{V}^t \\
&= \mathbf{R}(\theta)\mathbf{R}(-\phi)\mathbf{S}\mathbf{R}(\phi) \\
&= \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} \cos(-\phi) & -\sin(-\phi) \\ \sin(-\phi) & \cos(-\phi) \end{bmatrix} \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} \begin{bmatrix} \cos(\phi) & -\sin(\phi) \\ \sin(\phi) & \cos(\phi) \end{bmatrix}
\end{aligned}
\tag{19}
$$

$\mathbf{U}\mathbf{V}^t$ is an orthogonal matrix since $(\mathbf{U}\mathbf{V}^t)^t = (\mathbf{U}\mathbf{V}^t)^{-1}$. Therefore, without loss of generality, it can be written as a rotation matrix. Of the possible 6 DOFs (degrees of freedom)

of the affine transformation, the 4 DOFs in $\mathbf{A}$, i.e., (a, b, c, d) have been replaced with $(\theta, \lambda_1, \lambda_2, \phi)$.

The affine matrix $\mathbf{A}$ can therefore be viewed as a succession of the following 4 steps: (a) Rotation by angle ϕ, (b) scaling of λ_1 and λ_2 in the rotated x and y directions, (c) rotation by angle -ϕ which brings the scaled object back to its original orientation, and (d) rotation by angle θ. We now discuss how to convert between affine parameter representations.

1. Converting (a, b, c, d) to $(\theta, \lambda_1, \lambda_2, \phi)$. In several cases, the affine parameters are given in the form of (a, b, c, d). However, it is difficult to get a physical intuition when the parameterization is done in this form. In such cases, converting to $(\theta, \lambda_1, \lambda_2, \phi)$ helps in getting an insight into how the object of interest is being deformed. Also, for the particle filter, it is more intuitive to specify expected variances for $(\theta, \lambda_1, \lambda_2, \phi)$ than for (a, b, c, d). The first step here is to compute the SVD decomposition $\mathbf{A} = \mathbf{USV}^t$. The parameters $(\theta, \lambda_1, \lambda_2, \phi)$ are computed as follows,

$$\boxed{\begin{aligned} \phi &= \tan^{-1} \frac{v_{1,2}}{v_{1,1}} \\ \lambda_1 &= s_{1,1} \\ \lambda_2 &= s_{2,2} \\ \theta &= \tan^{-1} \frac{u_{2,1}v_{1,1}+u_{2,2}v_{1,2}}{u_{1,1}v_{1,1}+u_{1,2}v_{1,2}} \end{aligned}} \tag{20}$$

2. Converting $(\theta, \lambda_1, \lambda_2, \phi)$ to (a, b, c, d). In visual tracking, the initial target planar bounding region is more intuitively expressed in terms of $(\theta, \lambda_1, \lambda_2, \phi)$ than in terms of (a, b, c, d). However, the actual affine warp is more easily carried out using matrix multiplication for which we need (a, b, c, d). This can be done by multiplying out all the terms in Equation 19 to get

$$\boxed{\begin{aligned} a &= (\lambda_2)p + (\lambda_1)q \\ b &= (\lambda_2)s - (\lambda_1)r \\ c &= (\lambda_2)r - (\lambda_1)s \\ d &= (\lambda_2)q + (\lambda_1)p \end{aligned}} \tag{21}$$

where temporary variables p, q, r, s are computed from angles θ and ϕ using [1],

$$\begin{aligned} &\text{ccc} = \cos(\theta)\cos^2(\phi), \text{ccs} = \cos(\theta)\cos(\phi)\sin(\phi), \text{css} = \cos(\theta)\sin^2(\phi) \\ &\text{scc} = \sin(\theta)\cos^2(\phi), \text{scs} = \sin(\theta)\cos(\phi)\sin(\phi), \text{sss} = \sin(\theta)\sin^2(\phi) \\ &p = \text{css} - \text{scs}, q = \text{ccc} + \text{scs}, r = \text{ccs} + \text{sss}, s = \text{ccs} - \text{scc} \end{aligned} \tag{22}$$

During tracking, in the first frame, manually selected affine parameters and feature points are warped to an upright reference position as shown in Figure 28. During run-time, in every frame, the motion model (discussed next) generates several affine candidate parameter sets. Each of these sets is used to warp a zero-centered grid onto or around the object of interest. Bilinear interpolation is then used to compute pixel intensity values at the warped grid points as shown in Figure 29. For the example affine parameter set given in this figure, we see that the affinely warped region accurately samples the object of interest and minimizes inclusion of background pixels. Tracking error for a particular affine parameter set in a frame can be computed by warping the reference feature points from the first frame using this affine parameter set and computing rms error with ground truth feature points for that frame. We now discuss the motion model.

4.3 Model 2: Motion model

The motion model is a mathematical representation of the real or expected motion of the target of interest. Since tracking is in general an ill-posed problem, it is common to make assumptions about the motion to simplify motion modeling. Common assumptions such as stationary camera, coherent motion etc. are discussed in Chapter 3.

In this work, we make two assumptions about the target motion:

- Coherent motion. We assume that each part of the target moves together. The target can deform and warp but it does not break up into individual parts.
- Can be modeled with a gaussian distribution with fixed variance. We assume that the motion is brownian and can be modeled with a gaussian distrubution with a fixed variance.

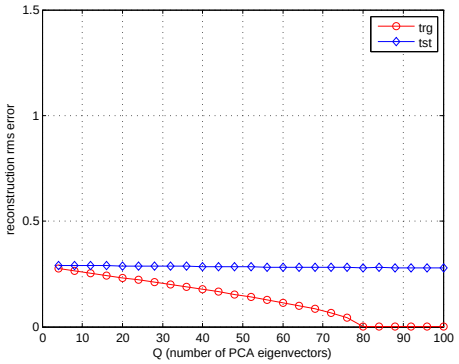
An advantage of not having an explicit motion model is that arbitrary camera and target motion are allowed. A disadvantage of this approach in the context of the particle filter is that particles need to be evaluated all around the current target position, rather than around a predicted target position in a certain direction. We are therefore unable to take advantage of the reduced spatial search-space that comes with a deterministic motion model.

At time t, the goal of the tracking process is to estimate the state vector $\mathbf{X}_t = (\theta, \lambda_1, \lambda_2, \phi, x, y)$. To keep our model as general as possible, all 6 components of the state vector are modeled as Gaussian random variables but with known variance which is specified in the first frame. However, in order to simplify sampling from the joint density, it is possible to use certain relaxation criteria such as Markovian dependence, or independence. We choose the latter to avoid MCMC sampling [113] and note that this method works well in practice. The target motion is therefore represented not in analytic form but as a 6x6 diagonal covariance matrix Σ_X centered at $\mathbf{X}_{t-1}$ in the previous frame. The elements on the diagonal represent variances of the affine parameters, $\sigma^2_\theta, \sigma^2_{\lambda_1}, \sigma^2_{\lambda_2}, \sigma^2_\phi, \sigma^2_x, \sigma^2_y$. For instance, for the x and y coordinates of the target at time t, the probability of the target position is given by,

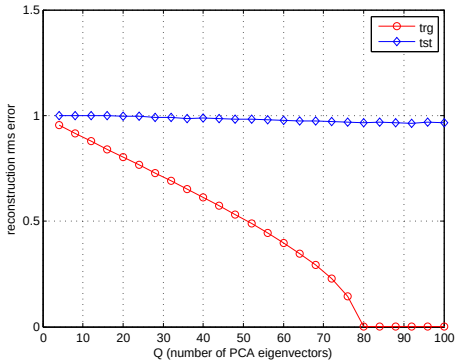
$$p(x_t|x_{t-1}) = \mathcal{N}(x_{t-1}, \sigma^2_x) \tag{23}$$

$$p(y_t|y_{t-1}) = \mathcal{N}(y_{t-1}, \sigma^2_y) \tag{24}$$

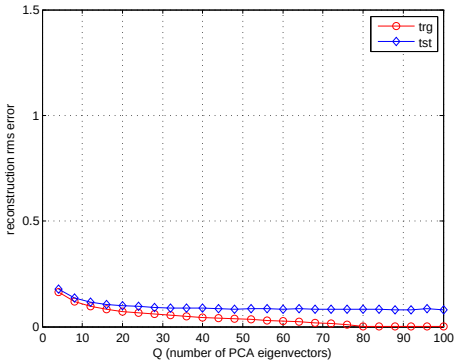
At every time step, predicted values are sampled from all 6 distributions. Each predicted set is used to warp a zero-centered grid onto or around the target of interest as explained in



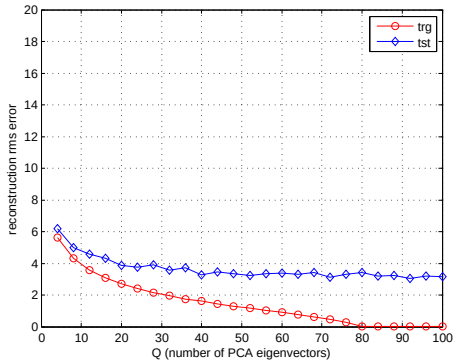
(a) Uniform random variable $U \sim [0, 1]$ in $\mathbb{R}^{1089}$, 100 realizations.



(b) Gaussian random variable $\mathcal{N} \sim (0, 1)$ in $\mathbb{R}^{1089}$, 100 realizations.



(c) Gauss-Markov random variable $\mathcal{N} \sim (0, 1)$ in $\mathbb{R}^{1089}$ with 0.9 correlation, 100 realizations.



(d) Dudek sequence, 33x33 ($\mathbb{R}^{1089}$) face snippets were extracted from the first 100 images.

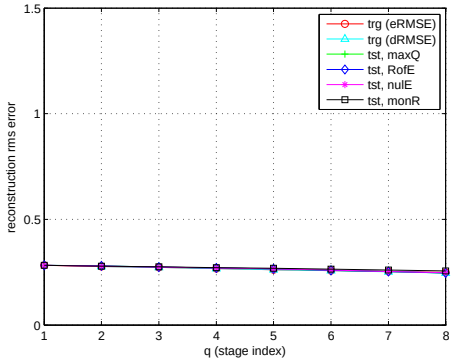
Figure 30. PCA, 100 training examples in $\mathbb{R}^{1089}$ were used for each of these experiments. Results were averaged over 10 cross-validation runs. For each run, 20% of the data, i.e., 20 examples were randomly picked for testing while the remaining 80 examples were used for training.

the previous section. Next, we discuss the inference model which is used to select the best set of affine candidates generated by the motion model.

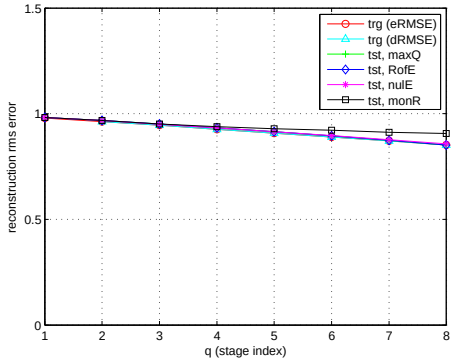
4.4 Model 3: Appearance model

Common appearance models include just the raw values of the pixel intensities [114, 79], pixel intensity distributions [78, 115, 116, 117], templates[118], active appearance models [119, 89], pixel intensity centroids [120] and subspace based methods [10, 8].

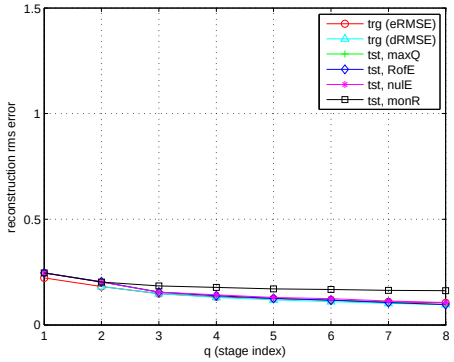
In order to understand appearance modeling, we conduct the following 4 experiments using PCA, RVQ and TSVQ to measure rms errors for target reconstruction:



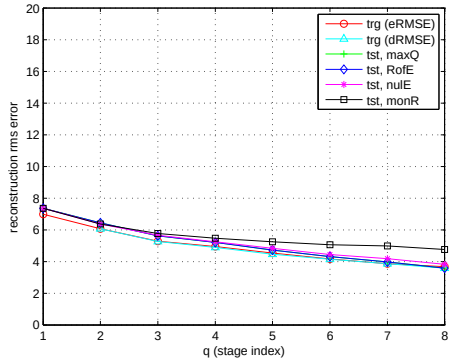
(a) Uniform random variable $U \sim [0, 1]$ in $\mathbb{R}^{1089}$, 100 realizations.



(b) Gaussian random variable $\mathcal{N} \sim (0, 1)$ in $\mathbb{R}^{1089}$, 100 realizations.



(c) Gauss-Markov random variable $\mathcal{N} \sim (0, 1)$ in $\mathbb{R}^{1089}$ with 0.9 correlation, 100 realizations.



(d) Dudek sequence, 33x33 ($\mathbb{R}^{1089}$) face snippets were extracted from the first 100 images.

Figure 31. RVQp, varying number of stages P with number of code-vectors per stage held constant at $M = 4$. 100 training examples in $\mathbb{R}^{1089}$ were used for each of these experiments. A single test example in $\mathbb{R}^{1089}$ was reconstructed.

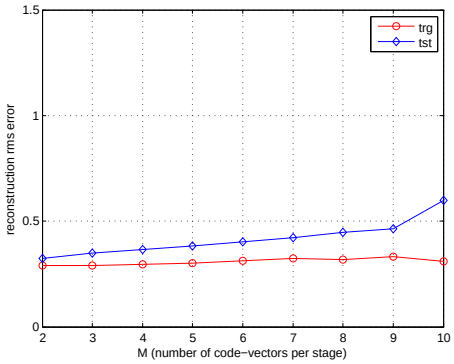
1. PCA: varying number of eigenvectors, Q.

2. RVQp: varying number of stages P for RVQ while holding the number of code-vectors per stage constant at $M = 4$.

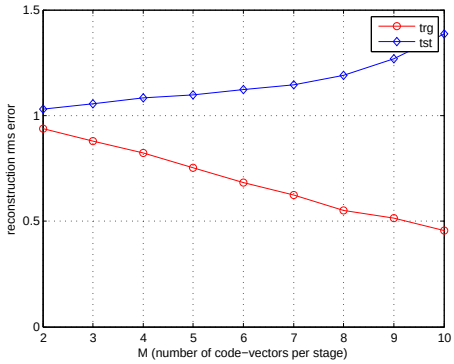
3. RVQm: varying number of code-vectors per stage M for RVQ while holding the number of stages constant at $P = 8$.

4. TSVQ: varying number of stages, P.

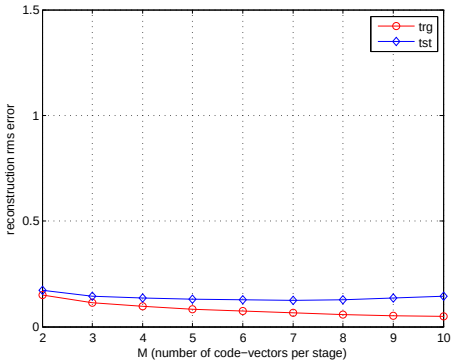
It is hoped that investigating reconstruction errors will aid in understanding the behavior of these various algorithms when used to model target appearance in tracking applications.



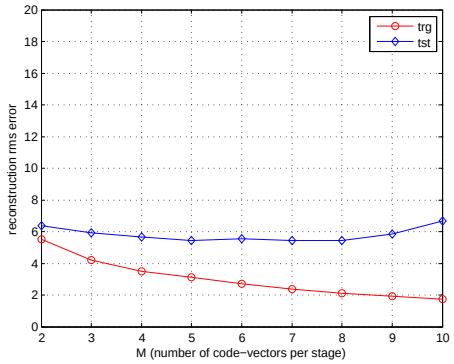
(a) Uniform random variable $U \sim [0, 1]$ in $\mathbb{R}^{1089}$, 100 realizations.



(b) Gaussian random variable $\mathcal{N} \sim (0, 1)$ in $\mathbb{R}^{1089}$, 100 realizations.



(c) Gauss-Markov random variable $\mathcal{N} \sim (0, 1)$ in $\mathbb{R}^{1089}$ with 0.9 correlation, 100 realizations.

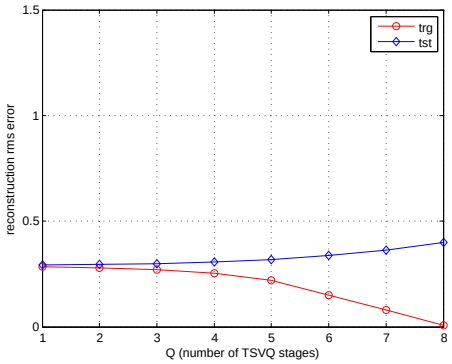


(d) Dudek sequence, 33x33 ($\mathbb{R}^{1089}$) face snippets were extracted from the first 100 images.

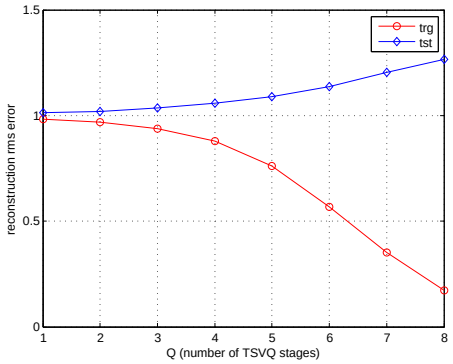
Figure 32. RVQm, experiments, varying number of code-vectors per stage M with number of stages held constant at $P = 8$. 100 training examples in $\mathbb{R}^{1089}$ were used for each of these experiments. Results were averaged over 10 cross-validation runs. For each run, 20% of the data, i.e., 20 examples were randomly picked for testing while the remaining 80 examples were used for training.

We use four datasets in $\mathbb{R}^{1089}$: (a) Uniform random variable, (b) Gaussian random variable, (c) Gauss-Markov random variable, and (d) images from the Dudek sequence. The reason for using $\mathbb{R}^{1089}$ is that our targets for all our tracking datasets are warped to a canonical size of 33-pixel height and 33-pixel width (33x33=1089). In all cases, we take 100 examples and split them up using an 80/20 rule, i.e. 80 training examples and 20 test examples. 10 cross-validation runs are used. In each cross-validation run, the training and test examples are picked randomly in the 80/20 ratio.

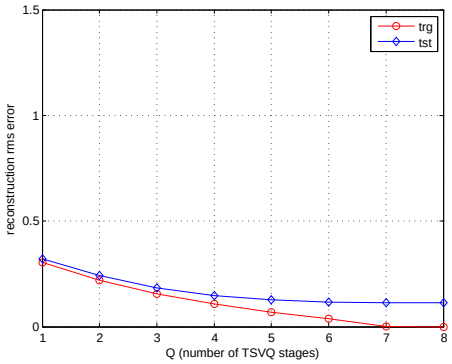
Results for PCA, RVQp, RVQm and TSVQ are shown in Figures 30, 31, 32 and 33 respectively. We make the following observations from these figures:



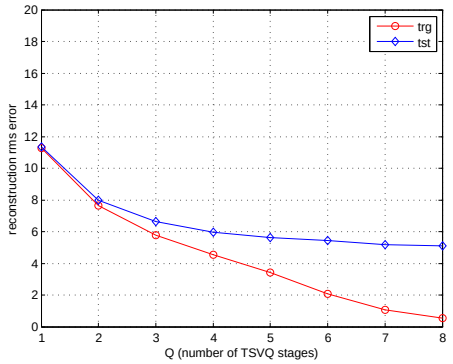
(a) Uniform random variable $U \sim [0, 1]$ in $\mathbb{R}^{1089}$, 100 realizations.



(b) Gaussian random variable $\mathcal{N} \sim(0, 1)$ in $\mathbb{R}^{1089}$, 100 realizations.



(c) Gauss-Markov random variable $\mathcal{N} \sim(0, 1)$ in $\mathbb{R}^{1089}$ with 0.9 correlation, 100 realizations.



(d) Dudek sequence, 33x33 ($\mathbb{R}^{1089}$) face snippets were extracted from the first 100 images.

Figure 33. TSVQ, 100 training examples in $\mathbb{R}^{1089}$ were used for each of these experiments. Results were averaged over 10 cross-validation runs. For each run, 20% of the data, i.e., 20 examples were randomly picked for testing while the remaining 80 examples were used for training.

1. **Training error**. Training error is always less than test error, as expected. Also, for each of the algorithms individually, we observe,

   - PCA: Monotonic decrease in rms reconstruction error with increasing Q. Training error becomes 0 when $Q = 80$ since there are 80 training examples.
   - RVQp: Monotonic decrease in rms reconstruction error with increasing P.
   - RVQm: Monotonic decrease or approximately constant rms reconstruction error with increasing M.
   - TSVQ: Monotonic decrease in rms reconstruction error with increasing P.

2. **Test error**.

- Statistically independent data: For the uniform and Gaussian random variables, test error for PCA and RVQp stays almost constant with increasing Q and P respectively. The reason is that PCA and RVQp use successive refinement when increasing Q and P respectively. Test error is therefore not expected to get better since it is not possible to better explain random data with increasing Q and P.

  For RVQm and TSVQ, test error increases with increasing M and P respectively. For TSVQ, increasing P controls its VC (Vapnik-Chervonenkis) dimension [121] and therefore its generalization ability [122]. It appears that increasing M in RVQm has a similar effect. The reason is that with $P = 1$, increasing M in RVQ is equivalent to increasing P in TSVQ. Increasing P in RVQ adds additional refinement to the equivalent code-vectors but M controls the overall general placement of RVQ code-vectors in the decision space $\mathbb{R}^D$. Therefore, in RVQ, it is M more than P that controls generalization ability. Therefore, when M in RVQm or P in TSVQ increase, their generalization ability decreases , leading to better explanation of training data, but with less ability to explain the test data well. For both RVQm and TSVQ, notice that when training error falls off sharply, test error increases sharply. Also, when training error drop is gradual, so is test error increase rate. This confirms over-training behavior. Also, RVQ increase or decrease rates are more gradual than TSVQ. The reason is that for RVQm, the number of equivalent code-vectors increase as $2^8, 3^8, 4^8, \ldots, 10^8$. Even for small values of M, the number of equivalent code-vectors is already quite large. For TSVQ, the terminal code-vectors increase as $2^1, 2^2, 2^3, \ldots, 2^8$ and therefore there is a rapid increase in the number of code-vectors from a very small value to a very large value.

  Finally, the rms reconstruction error is lower for uniform random data than

for Gaussian random data.[4] The reason is that the variance of the gaussian distribution is 1 while the variance of the uniform distribution with support [0, 1] is much lower at 1/12 [124].

- Statistically dependent data: For the Gauss-Markov and Dudek cases, all 4 algorithms display decreasing test error with increasing Q or P. The leveling off of the test error, or the "knee-point" [125], is visible in all cases.

In these experiments, we see that training error of PCA is in general better than RVQ. This is expected since PCA can achieve perfect reconstruction when Q comes close to the number of training examples N, $N << D$. Test errors however are comparable. RVQ has 2 knobs, P and M. In varying P, it acts like PCA in providing successive approximation. In varying M, it acts like TSVQ in changing its VC dimension, and therefore its generalization ability. Given this flexibility, it is expected that RVQ will perform well in our tracking framework.

4.5 Model 4: Observation model

An observation model $p(z_t|x_t)$ relates the state x_t at time t to the observation z_t at time t.

The observation model generates observations that will serve as candidates for the target, as shown in Figure 34 and then assigns scores to each candidate.

For PCA, it is assumed that an image $\mathbf{x}$ in $\mathbb{R}^D$ is probabilistically generated from a subspace $\mathbf{U}$ spanned by earlier observed images. The covariance matrix Σ of the input training images can be written as $\Sigma = \mathbf{U}\mathbf{\Lambda}\mathbf{V}^T$. Here $\mathbf{\Lambda}$ is the matrix of eigenvalues. The distribution is assumed to be Gaussian centered at $\boldsymbol{\mu}$. The probability of an image being generated under this distribution is inversely proportional to its distance from $\boldsymbol{\mu}$. This distance can be decomposed into two parts:

[4] It is may be tempting to explain this using an entropy argument. The uniform distribution has the maximum entropy among all continuous distributions with finite support $[a, b]$ while the Gaussian distribution has maximum entropy among all distributions with infinite support [123]. However, due to the difference in support, it is difficult to compare entropies.

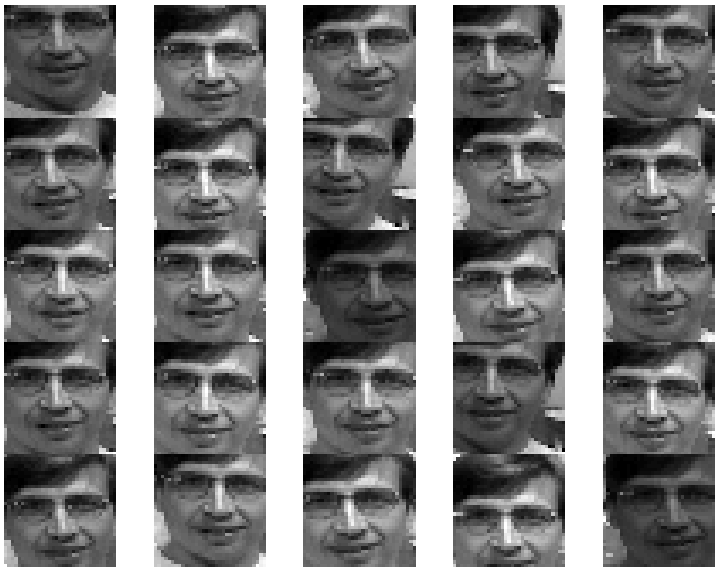


Figure 34. Different observations extracted from the frame at time t that will be evaluated to find the snippet that is best explained by the appearance model. The brightness changes in the various snippets are due to scaling.

1. DFFS (distance-from-feature-space): In a partial KL expansion using Q eigenvectors, the space spanned by these Q eigenvectors is given by $\mathbf{F}$[5] and the signal residual ϵ^2 is given by

$$\epsilon^2 = \|\tilde{\mathbf{x}}\|^2 - \sum_{i=1}^{Q} \mathbf{u}_i^2 = \sum_{i=Q+1}^{D} \mathbf{u}_i^2 \tag{25}$$

   where $\tilde{\mathbf{x}}$ is the mean removed input image and $\mathbf{u}_i$ are the eigenvectors of the covariance matrix estimate, $\mathbf{\Sigma} = \tilde{\mathbf{x}}\tilde{\mathbf{x}}^T$. This signal residual is referred to as DFFS.

2. DIFS (distance-in-feature-space): This is the component of $\mathbf{x}$ which lies in the feature space $\mathbf{F}$.

DIFS and DFFS are illustrated graphically in Figure 24. In a gaussian distribution, the probability of a data point $\mathbf{x}$ in $\mathbb{R}^D$ depends on the Mahalanobis distance d. The output of PCA, zero-centered $\tilde{\mathbf{y}}$ is decorrelated with variances along each dimension equal to the eigenvalues λ_i of the covariance matrix $\mathbf{\Sigma}$,

[5]We use $\mathbf{U}$ interchangeably with $\mathbf{F}$ here. Whereas the notation $\mathbf{U}$ is more widely used to represent a PCA eigenspace, we use $\mathbf{F}$ to remain compatible with Figure 24 taken from [10].

$$
\begin{aligned}
d &= (\mathbf{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu}) \\
&= \tilde{\mathbf{x}}^T\boldsymbol{\Sigma}^{-1}\tilde{\mathbf{x}} \\
&= \tilde{\mathbf{x}}^T(\mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^T)^{-1}\tilde{\mathbf{x}} \\
&= \tilde{\mathbf{x}}^T(\mathbf{U}\boldsymbol{\Lambda}^{-1}\mathbf{U}^{-1})\tilde{\mathbf{x}} \\
&= (\mathbf{U}^T\tilde{\mathbf{x}})^T\boldsymbol{\Lambda}^{-1}(\mathbf{U}^T\tilde{\mathbf{x}}) \\
&= \tilde{\mathbf{y}}^T\boldsymbol{\Lambda}^{-1}\tilde{\mathbf{y}} \\
&= \sum_{d=1}^{D}\frac{\tilde{y}_i}{\lambda_i} \\
&= \sum_{d=1}^{Q}\frac{\tilde{y}_i^2}{\lambda_i} + \sum_{d=Q+1}^{D}\frac{\tilde{y}_i^2}{\lambda_i}, \qquad \left(\text{DFFS} = \text{recon. error} = \sum_{i=1}^{D} e_i^2 = \sum_{i=1}^{D}\tilde{x}_i^2 - \sum_{i=1}^{Q}\tilde{y}_i^2 = \sum_{d=Q+1}^{D}\tilde{y}_i^2\right) \\
&= \sum_{d=1}^{Q}\frac{\tilde{y}_i^2}{\lambda_i} + \frac{1}{\rho}\sum_{i=1}^{D} e_i^2 \qquad \left(\rho^* = \frac{1}{D-q}\sum_{i=q+1}^{D}\lambda_i\right)
\end{aligned}
\tag{26}
$$

This formulation, first presented in [10] shows that the first term in the sum is the DIFS term in Figure 24 while the second term corresponds to DFFS. With this formulation, PCA can be used in a probabilistic framework since the error of a test vector $\mathbf{x}$ now also depends on its distance from the mean of the data. However, as mentioned in [1], it is difficult to weight these two terms. In this work, we therefore only use DIFS so that our approach does not rely on finding different weights for different datasets.

As mentioned earlier, VQ, like PCA, does not define a proper density in the observation space [126]. However, it is common to assign a probability measure to a new data point in proportion to the distance of the closest centroid [126],

$$
p(\mathbf{x}_i|\boldsymbol{\mu}_k) = \frac{e^{-\left((\mathbf{x}_i-\boldsymbol{\mu}_k)^T(\mathbf{x}_i-\boldsymbol{\mu}_k)+\lambda(P_{\max}-P_i)\right)}}{\sum_{i=1}^{N} e^{-\left((\mathbf{x}_i-\boldsymbol{\mu}_k)^T(\mathbf{x}_i-\boldsymbol{\mu}_k)+\lambda(P_{\max}-P_i)\right)}} \tag{27}
$$

Here, $\boldsymbol{\mu}_k$ is the closest code-vector to test data-point to $\mathbf{x}_i$, $P_{\max}$ is the number of stages in the codebook, P_i is the number of stages required to decode $\mathbf{x}_i$, and λ is a regularization parameter. We use 4 different RVQ methods to compute which $\boldsymbol{\mu}_k$ input data-point $\mathbf{x}_i$ maps to,

1. maxP: In this method, RVQ decoding is carried out so that maximum stages P are used.

2. RofE: In this method, realm of experience coding is used. In other words, a test vector is decoded such that the decode path traversed belongs to the set of training decode paths.

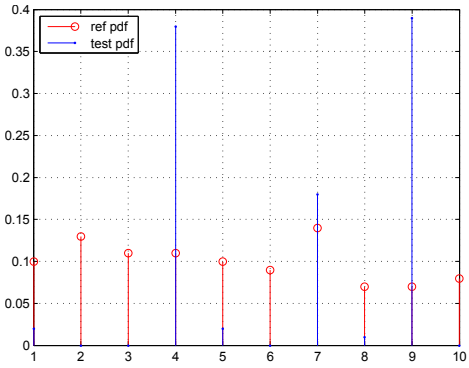
3. nulE: In this method, null encoding is used. Reconstruction rms error is checked at every stage. If at any stage, rms error is not reduced, that stage is skipped.

4. monR: In this method, monotonic rms error is a condition. If this condition is not met, decoding stops.

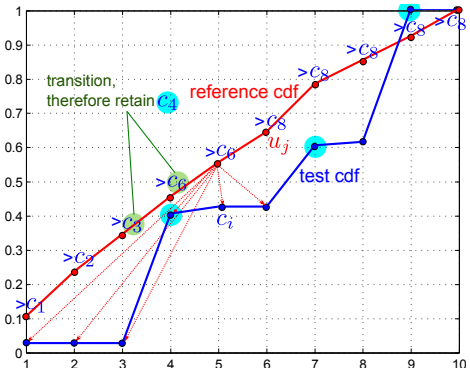
In our tracking framework, we use all 4 methods above and compare their performance.

4.6 Model 5: Inference model

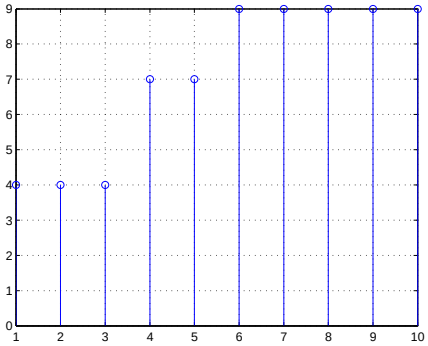
The inference model makes the final decision of which candidate snippet to pick as the target. Our inference model makes no assumption of linearity or Gaussianity. What this means is that we do not assume that the motion or observation models are linear, nor do we assume that the likelihood of finding a target at a particular location has a Gaussian distribution. Morever, we would like to keep a history of possible target candidate states, 600 in this case, so that soft decisions can be made about the target state at each frame. In other words, at every frame, even though we make a decision as to which particular snippet best represents the target, we acknowledge that this decision could be erroneous and therefore we propagate other candidates through time. This allows us to revisit candidate snippets that were not picked in previous frames as the target estimate but that could still have a high probability of being the correct snippet. Also, we do not want our hypotheses to grow with time. Keeping all this in mind, we base our inference model on the sequential Monte Carlo (SMC) filter, i.e., the particle filter [12].



(a) Reference (uniform) density and test PDF.



(b) Comparing CDFs.



(c) Particles 4, 7 and 9 are picked repeatedly since they have higher weight.

Figure 35. Particle filter, resampling.

As mentioned earlier, the weights of the target candidate snippets are computed in the observation model. An example scenario using only 10 candidate snippets, i.e., particles, at time t is given in Figure 35. The test pdf in this figure is generated by normalizing the weights of the target candidate snippets. In the first step, the inference model picks the 9th snippet as the target estimate since it has highest weight. In the second step, the inference model resamples the test pdf to remove snippets that have low weight and to repeat snippets that have high weight. To do this, a reference uniform density is generated. The CDFs of these 2 densites are then compared. Only those snippets are kept and repeated that have high weight as shown in the figure. The new resampled density is then passed to the motion model to generate candidate snippets in the next frame.

Having explained all 5 models that constitute our tracking framework, we now apply

our 6 trackers, PCA, TSVQ, maxP, RofE, nulE and monR to several image datasets. All 6 trackers share exactly the same representation, motion and inference models. However, each has its own appearance and observation models.

We now present tracking results in the next chapter.

CHAPTER 5

RESULTS

In this chapter, we present tracking error results for 6 different trackers, PCA-based, TSVQ-based and 4 RVQ-based trackers, maxP, RofE, nulE, and monR. This chapter is organized as follows:

1. Datasets. We first present some details about the datasets used.

2. Results. Tracking results are presented in 5 figures, Figures 37, 38, 39, 40 and 41. Our conclusions presented in the next chapter are based on the information presented in these 5 figures. These figures are derived from detailed experimental results for PCA, TSVQ, maxP, RofE, nulE and monR based tracking given in 6 figures, Figures 43, 44, 45, 46 47 and 48 respectively in Appendix 7.2.

We begin by introducing the datasets we used in our work, and then present our results.

5.1 Datasets

All trackers were run on 6 publicly available datasets, Dudek, davidin300, sylv, fish, car4 and car11. These datasets can be downloaded from [1]. See Figure 42 in Appendix 7.1 for snapshots of images from each of these datasets at 100 frame intervals. Tracking error was measured on each of these datasets using the error between ground truth feature points and estimated feature points as shown in Figure 36 for the Dudek sequence.

We begin by making some observations about each of these datasets. This information is also presented in summarized form in Table 2:

1. Dudek, davidin300. These sequences consist of indoors tracking of a human face through lighting, pose and expression changes with structured noise (putting on and taking off glasses). In addition, the Dudek sequence has temporary occlusions and sudden motion. These two sequences can be considered to be the most challenging

Table 2. Publicly available datasets used for RVQ tracking [1]. For lighting change, a value of 1 indicates mild lighting change while a value of 5 indicates severe lighting change. Structured noise includes taking off and putting on eye-glasses.

	Dudek	**davidin300**	**sylv**	**fish**	**car4**	**car11**
Scenario	Indoors	Indoors	Indoors	Indoors	Outdoors	Outdoors
Time of day	N/A	N/A	N/A	N/A	day	night
Target of interest	face	face	toy	inanimate object	vehicle	vehicle
Rigid target	no	no	no	yes	yes	yes
Lighting change	2	3	2	5	2	1
Sudden target motion	yes	yes	no	yes	no	no
Structured noise	yes	yes	no	no	no	no
Camera motion	yes	yes	yes	yes	yes	yes
Pose change	yes	yes	yes	no	yes	yes
Expression change	yes	yes	N/A	N/A	N/A	N/A
Temporary occlusion	yes	yes	no	no	no	no

datasets since they both have several different forms of noise. A significant form of noise is blur due to sudden motion. Additionally, these two sequences consist of tracking a face. In related areas related to face processing, such as face recognition, it has been shown that 40 eigenfaces can be used to reconstruct a face with 3% error [127]. However, performance levels off at about 25 principal components, or 45 principal components if the first 3 principal components are dropped [128]. The reason for dropping 3 principal components is that [129] showed that for a fixed viewpoint, images of a Lambertian surface[1] under varying lighting conditions lie in a 3D linear subspace of the high-dimensional image space. Although the first 3 principal components account for lighting changes in faces, these components are unlikely to only account for lighting variation and removing them may result in loss of important information [128]. Therefore, we do not remove any principal components in our PCA based tracker. However, unlike the face recognition case, our tracking performance does not keep increasing till 20 or more eigenvectors. The reason is that face alignment is noisy in tracking applications. It appears that in these two sequences which have large pose changes, the first few eigenvectors are able to capture the linear dependencies in the slightly shifted faces. After that, the later

[1] A Lambertian surface, or informally a matte surface, is a surface that has constant BRDF (bidirectional reflectance distribution function). BRDF has been explained earlier in the chapter.

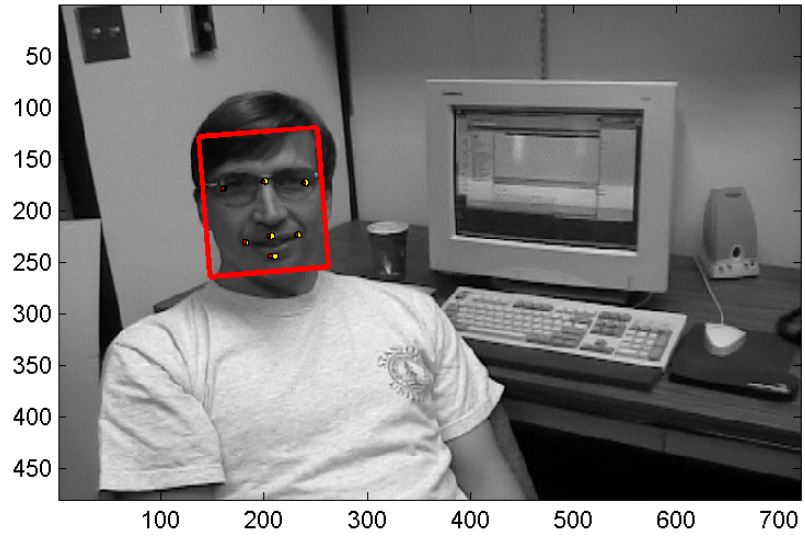


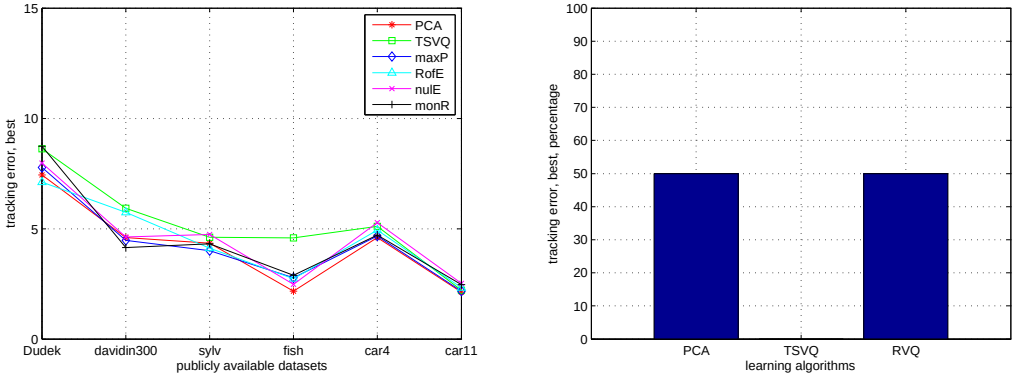
Figure 36. Computing tracking error. The larger yellow circles indicate ground truth feature points. The smaller red circles indicate estimated feature points. Tracking error is computed using the rms error between the ground truth feature points and the estimated feature points. In this particular frame, the tracking error is 2.57.

eigenvectors explain the residual noise. This can lead to decreased tracking performance since reconstructions using an eigenspace that partially explains noise will be noisy. Noisy reconstructions will get inaccurate DFFS (distance-from-feature-space) scores, which in turn will cause incorrect weighting for particle filter candidates in the tracking process. This will lead to larger tracking error.

2. Fish. This sequence consists of tracking a fish shaped wooden structure that is placed on a table. In this indoors sequence, the lights are turned on and off resulting in large and abrupt lighting changes. Moreover, the camera is shaken violently throughout the sequence.

3. Sylv, car4 and car11. The sylv sequence consists of indoor tracking of a plush toy with some lighting variation and some pose changes. The car4 and car11 sequences consist of outdoor tracking of vehicles during the day and night respectively. The car4 sequence has one period of large lighting change when it moves under a bridge shadow. The pose variation is gradual as the car changes lanes. The car11 sequence

	PCA	TSVQ	maxP	RofE	nulE	monR
Dudek	7.44	8.62	7.78	7.11	7.97	8.73
davidin300	4.60	5.93	4.47	5.74	4.63	4.15
sylv	4.34	4.61	4.00	4.12	4.74	4.31
fish	2.17	4.59	2.78	2.73	2.48	2.89
car4	4.60	5.11	4.67	4.93	5.28	4.71
car11	2.13	2.21	2.17	2.33	2.52	2.47
% best	50.00	0.00	16.67	16.67	0.00	16.67



(a) Best tracking error for each algorithm.

(b) %age of datasets over which best tracking error is achieved over all parameters.

Figure 37. Tracking results (1 of 5), comparison of best tracking performance. PCA give best performance for half the datasets, i.e. 3 datasets, while RVQ gives best performance for the other half.

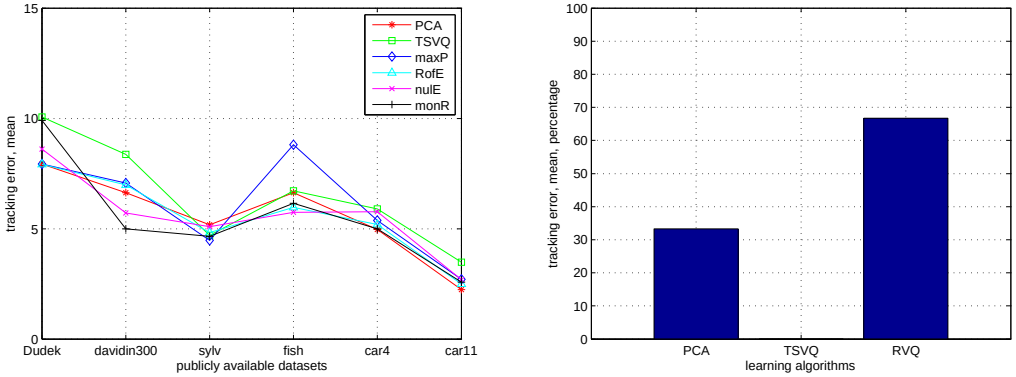
also has less variation in lighting and pose as compared to some of the other sequences.

We now present our results based on 5 metrics, best possible performance for each algorithm over all its parameters (Figure 37), mean performance for each algorithm (Figure 38), tracking performance if only 16 (Figure 39) or 32 (Figure 40) eigenvectors or code-vectors are stored in memory, and finally, mean tracking performance over all datasets for each algorithm (Figure 41).

5.2 Best performance

We start with Figure 37. In this figure, we plot best possible tracking performance for each algorithm. For PCA, this means the best possible performance attained for each of the datasets for number of eigenvectors Q=8, 16 and 32. For TSVQ, best possible performance

	PCA	TSVQ	maxP	RofE	nulE	monR
Dudek	7.93	10.07	7.93	7.91	8.60	9.90
davidin300	6.63	8.37	7.07	6.99	5.72	4.99
sylv	5.18	4.70	4.47	4.83	5.10	4.66
fish	6.63	6.71	8.81	5.97	5.74	6.15
car4	4.97	5.90	5.38	5.19	5.77	4.99
car11	2.24	3.48	2.70	2.49	2.69	2.58
% best	33.33	0.00	16.67	16.67	16.67	16.67



(a) Mean tracking error for each algorithm.

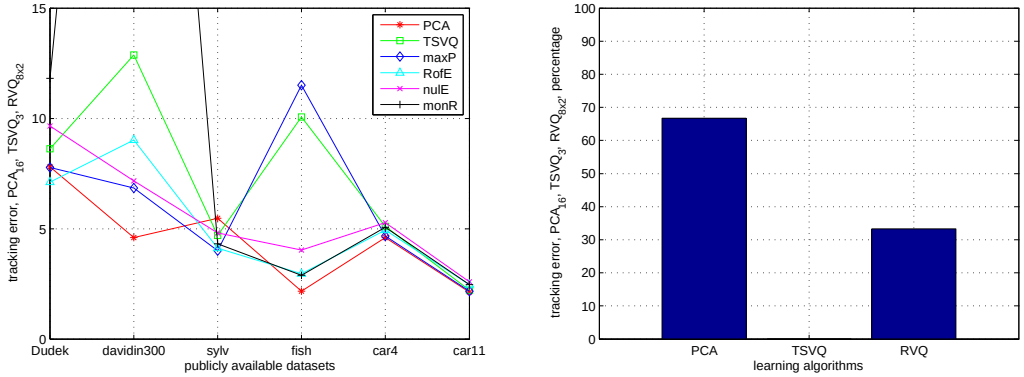
(b) %age of datasets over which best mean tracking error is achieved over all parameters.

Figure 38. Tracking results (2 of 5), comparison of mean tracking performance. RVQ performs better over twice as many datasets as PCA.

for each dataset is over number of stages P=3, 4 and 5. For maxP, RofE, nulE and monR, best possible performance for each dataset is over number of stages P and number of code-vectors M, PxM=8x2, 8x4 and 8x8. The reason for plotting performance for each dataset separately is that each dataset represents a different distribution and we would like to gauge performance for each algorithm over the different distributions. We see that performance for PCA and all 4 RVQ based algorithms is very close while TSVQ tracking error is highest in many cases.

PCA performs best in the fish, car4 and car11 sequences while RVQ performs best in the remaining three datasets, Dudek, davidin300 and sylv. TSVQ does not perform best in any sequence. Note that the performance difference between PCA and RVQ in the car4 and car11 sequences is negligible. Recall that car4 and car11 are relatively benign datasets with little variation in pose and lighting. The fish sequence has sudden motion as well as sudden

	PCA	TSVQ	maxP	RofE	nulE	monR
Dudek	7.81	8.62	7.78	7.11	9.65	11.81
davidin300	4.60	12.88	6.84	9.02	7.17	50.00
sylv	5.47	4.70	4.00	4.12	4.81	4.31
fish	2.17	10.07	11.50	2.96	4.03	2.89
car4	4.60	5.11	4.67	4.93	5.28	5.07
car11	2.13	2.21	2.17	2.47	2.59	2.47
% best	66.67	0.00	16.67	16.67	0.00	0.00



(a) Tracking error for each algorithm with 16 eigenvectors/code-vectors stored in memory.

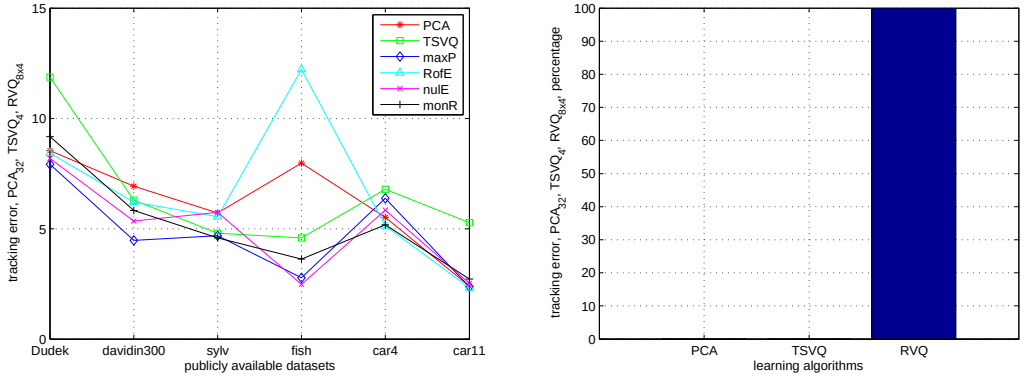
(b) %age of datasets over which best tracking error is achieved with 16 eigenvectors/code-vectors stored in memory.

Figure 39. Tracking results (3 of 5), comparison of tracking performance if 16 eigenvectors/code-vectors are stored in memory. PCA performs better over twice as many datasets as RVQ.

global lighting changes. Since global lighting change induces linear correlation in the data, it makes sense that PCA does well in this sequence. The reason is that global illumination changes move the illuminated object within the modeled PCA subspace [127].

RVQ performs best over the Dudek, davidin300 and sylv sequences. All 3 of these sequences have moderate lighting changes while Dudek and davidin300 have several forms of noise as discussed earlier. For Dudek, RofE does best. The reason is that in the presence of uncertainties, RofE holds tight to what has already been modeled and is resistant to accepting sudden changes in the underlying distribution. It is therefore better able to handle blur and other forms of noise that do not exist in the training data. On the other extreme is monR which greedily attempts to minimize reconstruction error. Out of all RVQ methods, this method performs worse, but even then, not by much. Second best performance is for maxP which is again not a greedy method. Third best performance is for nulE which is

	PCA	TSVQ	maxP	RofE	nulE	monR
Dudek	8.54	11.87	7.92	8.43	8.19	9.17
davidin300	6.93	6.29	4.47	6.21	5.35	5.83
sylv	5.72	4.80	4.68	5.54	5.74	4.58
fish	7.98	4.59	2.78	12.22	2.48	3.62
car4	5.52	6.79	6.38	5.14	5.84	5.18
car11	2.39	5.28	2.36	2.33	2.52	2.72
% best	0.00	0.00	33.33	33.33	16.67	16.67



(a) Tracking error for each algorithm with 32 eigenvectors/code-vectors stored in memory.

(b) %age of datasets over which best tracking error is achieved with 32 eigenvectors/code-vectors stored in memory.

Figure 40. Tracking results (4 of 5), comparison of tracking performance if 32 eigenvectors/code-vectors are stored in memory. RVQ performs the best over all datasets.

also a greedy method but less so than monR.

5.3 Mean performance over parameters

We now turn to Figure 38. In this figure, mean performance over all parameters is shown. It may be noted that monR loses track in one instance. That instance is not factored into the means since it is not clear how penalize a lost track when performing mean computations. Here, we see that RVQ performs best 66.7% of the time. This time, in addition to Dudek, davindin300 and sylv, RVQ performs better than PCA in the fish sequence as well. The reason for this is that PCA is unable to track the fish sequence well when it has too few, i.e., 8 eigenvectors or when it has too many, i.e., 32 eigenvectors. In the 8 eigenvector case, the subspace does not have enough dimensions to model lighting changes well. Even though it has been shown, as mentioned earlier, that only 3 eigenvectors are needed to model lighting

8	16	32	mean
6.15	4.46	6.18	5.60

(a) PCA.

3	4	5	mean
7.26	6.60	5.74	6.54

(b) TSVQ.

8x2	8x4	8x8	mean
6.16	4.76	7.25	6.06

(c) maxP.

8x2	8x4	8x8	mean
5.10	6.64	4.94	5.56

(d) RofE.

8x2	8x4	8x8	mean
5.59	5.02	6.20	5.60

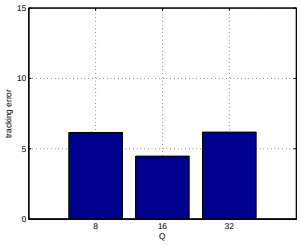
(e) nulE.

8x2	8x4	8x8	mean
5.31	5.18	6.19	5.56

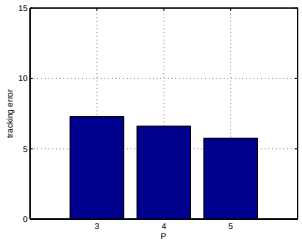
(f) monR.

8x2	8x4	8x8
5.54	5.40	6.15

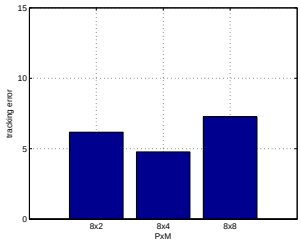
(g) maxP, RofE, nulE, monR.



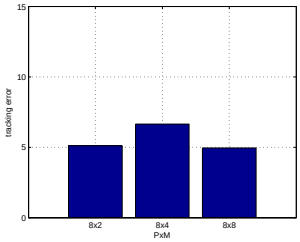
(h) PCA



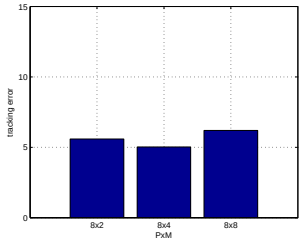
(i) TSVQ.



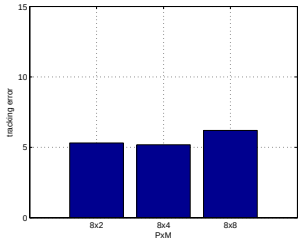
(j) maxP.



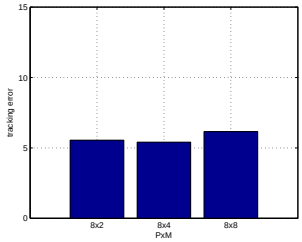
(k) RofE.



(l) nulE.



(m) monR.



(n) maxP, RofE, nulE, monR.

Figure 41. Tracking results (5 of 5), comparison of tracking performance as parameters for each algorithm are varied. In (d), we see that over all RVQ algorithms, RofE has best mean performance. In (g) it is clear that the best RVQ configuration is 8x4.

changes [127], in practice this does not hold due to shadowing and specularities [128]. For too many eigenvectors, over-fitting is an issue. For $Q = 16$, PCA performs best and that is why it had best possible performance. However, when it comes to means, all 4 RVQ

parameters are able to outperform PCA in mean performance.

5.4 Memory = 16 vectors

In Figure 39, we hold the number of eigenvectors for PCA or codevectors for TSVQ and RVQ constant at 16 (actually 14 for TSVQ but we ignore this slight difference). In these figures, we see that PCA outperforms RVQ for 16 vectors.

5.5 Memory = 32 vectors

In Figure 40, we hold the number of eigenvectors for PCA or codevectors for TSVQ and RVQ constant at 32 (actually 30 for TSVQ but we ignore this slight difference). In these figures, we see that RVQ completely outperforms PCA for 32 vectors. The reason is that at 8x4, RVQ now has enough capacity to explain the underlying distributions, and is better able to do so than PCA or TSVQ.

5.6 Mean performance over datasets

Finally, in Figure 41, we plot mean tracking performance over all datasets for each algorithm. Here we see that PCA performs best for Q = 16, while both RofE and monR have best mean performance over all parameters and over all datasets. Moreover, over all RVQ configurations, 8x4 performs best when averaged over all datasets.

In this figure, for 3 parameters per algorithm, Q=8, 16, 32 for PCA, P=3, 4, 5 for TSVQ and PxM = 8x2, 8x4, 8x8 for RVQ, there are 4 possible outcomes listed below. Of these, the first 3 are to be expected. The fourth however requires further scrutiny.

1. Monotonically increasing error. This would mean that the degrees of freedom (DoF) in the learning algorithm, such as PCA, TSVQ or RVQ, model the underlying distribution well with low DoFs and adding DoFs is leading to over-training. We do not see this performance in any case since we start with low DoFs for each algorithm. We got an initial estimate of how many DoFs to use in our experiments on appearance

modeling as shown in Figures 30, 31 32 and 33.

2. Monotonically decreasing error. This happens for TSVQ. This means that adding more stages to TSVQ may increase performance and that TSVQ has not yet achieved optimum performance. In our case, we use 3, 4 and 5 stages to keep the total number of stored code-vectors in TSVQ close to the stored code-vectors for RVQ and stored eigenvectors for PCA. This also shows that PCA and RVQ are able to achieve optimum performance with less required memory storage than TSVQ.

3. Decreasing error followed by increasing error. We see this performance for PCA, maxP, nulE and monR. This is a sign that the correct number of DoFs were chosen and that when error is minimum, the algorithm now has enough capacity to model the underlying distribution, but without over-fitting.

4. Increasing error followed by decreasing error. We see this in one case, RofE, and in some cases in TSVQ in Figure 44. To see this, consider the example of K=2, 4 and 8 code-vectors in $\mathbb{R}$ uniformly spaced on the inteval [0,7]. For K=2, the code-vectors are 2.33 and 4.66. For the K=4 case, the code-vectors are 1.4, 2.8, 4.2 and 5.6. For the K=8 case, the code-vectors are $\{0, 1, 2, 3, \ldots, 7\}$. In regions around 2.33 and 4.6, there are certain contiguous regions where the reconstruction error is greatest for K=4. This shows that although in general, one would not expect reconstruction error to be lowest for an intermediate number of code-vectors K, it is possible for a test vector to have highest reconstruction error for an intermediate K. If such an error occurs during tracking, then in certain cases, it may not be possible to recover from this error. See Figures 49, 50 and 51 in Appendix 7.3 for an example of such a scenario for TSVQ while the target is moving quickly and the tracker is unable to recover from a single incorrect decision. In learning based tracking, wrong decisions can be costly since there is no supervised learning mechanism to correct wrong labels.

Having presented our results, we now present our conclusions in the next chapter.

CHAPTER 6

CONCLUSIONS

In this work, we demonstrated successful application of RVQ for visual tracking over a variety of datasets, and compared our results with PCA and TSVQ based tracking. We have based our design on a well-known method for visual tracking [1]. The advantage of using an existing tracking framework is that it allows our newer RVQ based tracking method to be compared more easily with existing methods in the literature.

Based on our work, we draw the following conclusions:

1. Performance comparison. We chose five metrics to compare PCA, TSVQ and RVQ. A sixth metric is added which counts number of lost tracks.

   (a) Best possible performance. PCA and RVQ performed best in half the times each. TSVQ never performed best. However, of the 3 times that PCA performed best, in 2 cases, the performance difference was not significantly better than RVQ. Moreover, and perhaps more importantly, RVQ performed best in the two most challenging datasets, Dudek and davidin300 since they both have multiple sources of noise.

   (b) Mean performance over parameters. Here, RVQ performed best in twice the number of scenarios as PCA. TSVQ had the worst mean performance.

   (c) Memory cost=16 vectors. Here PCA performed best in twice the number of scenarios as RVQ .

   (d) Memory cost=32 vectors. Here RVQ completely outperformed PCA and TSVQ. This is understandable since the capacity of RVQ to explain an underlying distribution grows exponentially as M^P. At 8x4, RVQ has enough capacity to track well without over-fitting.

(e) Mean performance over datasets. Here, PCA with $Q = 16$ has best performance. However, if performance is averaged over datasets and parameters, then RofE and monR perform best. Of the different RVQ parameters, 8x4 has best performance.

(f) Lost tracks. There was only one lost track for monR. This is understandable since monR is a greedy approach. The lost track was in davidin300 which is a challenging dataset.

2. Target alignment. In tracking scenarios, accurate alignment of targets is difficult. In the case of PCA for instance, it has been mentioned earlier that between 25 to 45 eigenvectors can be used for accurate face reconstruction [128]. In our case, for the Dudek sequence that has faces, PCA with 16 eigenvectors was able to capture the linear dependence between slightly shifted versions of the same target since slight shifts still preserve correlation. However, as the number of eigenvectors increased further, the additional eigenvectors explained noise in the data. This scenario can lead to noisy reconstructions and subsequent noisy weighting for target candidates. When the noisy target candidate that is best explained by the PCA subspace is then added to the training set to update the PCA subspace, the resulting subspace will be noisy which will further increase the chances of noisy reconstructions.

Overall, PCA and RVQ outperform TSVQ completely. Between PCA and RVQ, RVQ outperforms PCA in more areas. It appears that in a tracking scenario, it is more useful to model a target by the means computed from its instances than by assuming that its observations are generated from an underlying subspace. In other words, a data dependent approach like RVQ more accurately models the target than an approach that attempts to build a subspace from limited data.

Moreover, in a tracking scenario, it is desirable to try different algorithms to get a feel for the dynamics of the underlying distribution. Specifically, it is desired to understand how

many DoFs are needed for a given algorithm to explain the distribution. Our experiments show that averaged over all distributions and over all algorithms and their parameters, 8x4 RofE performs the best. The reason is that 8x4 has a moderate VC dimension and that RofE is resistant to noise since it will penalize candidate snippets that have not appeared before. In this sense, it allows the RVQ codebook to adapt gradually to the changing underlying distribution as tracking progresses over time.

Our next step is to explore multiple targets, multi-spectral inputs, comparison with non-linear manifold learning methods such as LLE (locally linear embedding) and MDS (multidimensional scaling), using higher stage refinement for RVQ, and an investigation into the relation of RVQ with PCA using the subspace based approach given in [130].

CHAPTER 7

APPENDICES

The appendices that follow contain sample images from the datasets used in this work as well as detailed tracking plots for the interested reader.

7.1 Datasets

(a) Dudek.

(b) Davidin300.

(c) Sylv.

(d) Fish.

(e) Car4.

(f) Car11.

Figure 42. Publicly available tracking sequences downloadable from [1].

7.2 Tracking error plots

The following 6 pages contain 6 figures corresponding to tracking error plots for PCA, TSVQ, maxP, RofE, nulE and monR based tracking. Each figure comprises a table and 4 plots. Each entry in a table represents tracking error temporally averaged over the frames of a dataset (most of the datasets have more than 500 images). The entries in a table are visualized in the accompanying 4 plots. The plots show tracking error for different parameter values and their averages, and tracking error for different datasets and their averages.

7.2.1 PCA

	Q=8	Q=16	Q=32
1. Dudek	7.44	7.81	8.54
2. davidin300	8.36	4.60	6.93
3. sylv	4.34	5.47	5.72
4. fish	9.75	2.17	7.98
5. car4	4.79	4.60	5.52
6. car11	2.21	2.13	2.39
mean	6.15	4.46	6.18

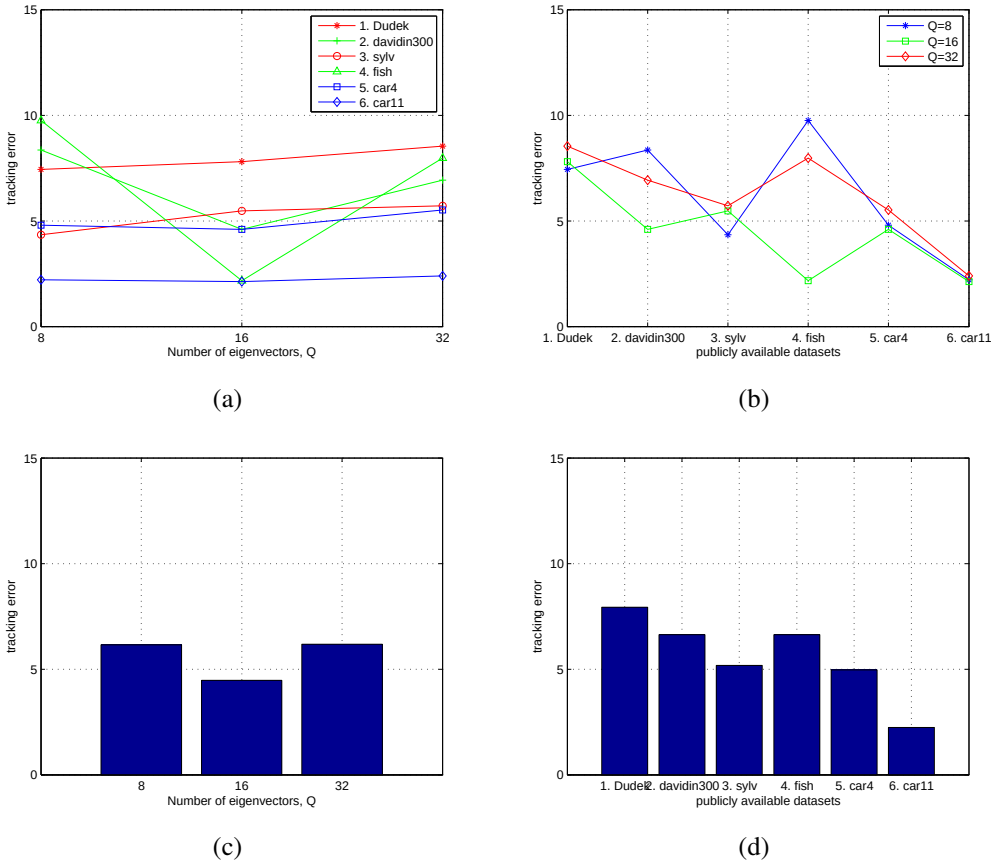


Figure 43. Tracking error for PCA based tracking for different number of eigenvectors Q for 6 different publicly available datasets.

7.2.2 TSVQ

	P=3	P=4	P=5
1. Dudek	8.62	11.87	9.71
2. davidin300	12.88	6.29	5.93
3. sylv	4.70	4.80	4.61
4. fish	10.07	4.59	5.47
5. car4	5.11	6.79	5.80
6. car11	2.21	5.28	2.94
mean	7.26	6.60	5.74

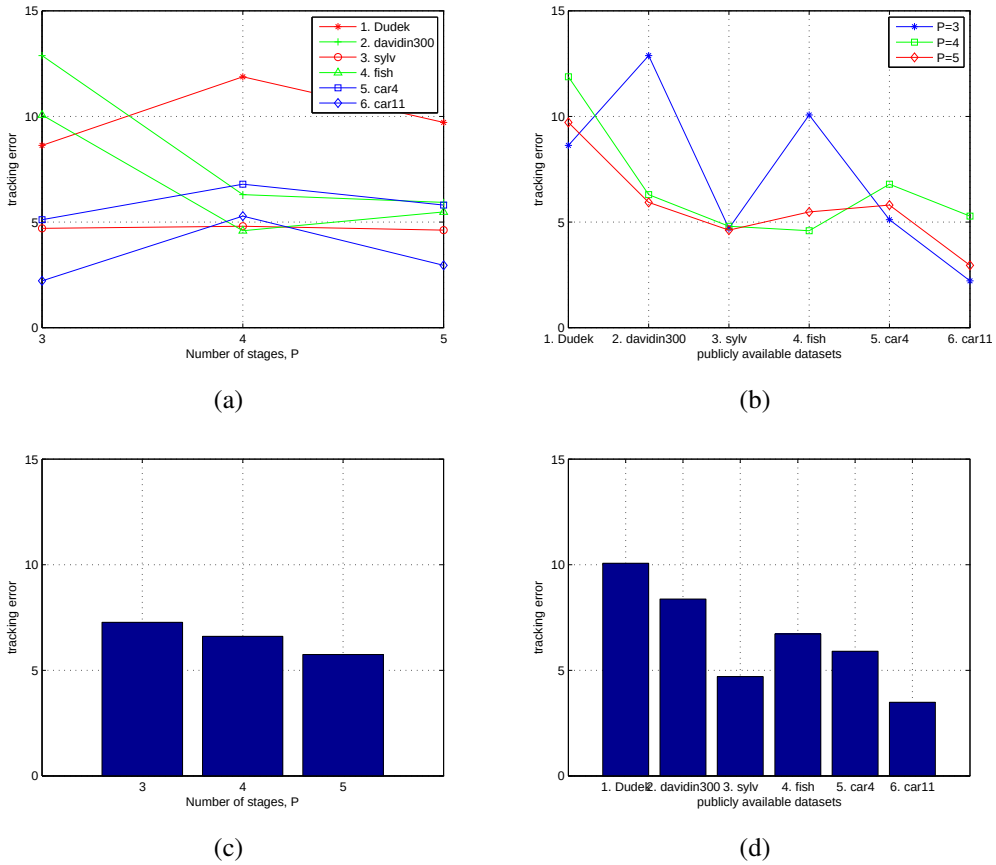


Figure 44. Tracking error for binary balanced-tree-TSVQ based tracking for different number of stages P for 6 different publicly available datasets.

7.2.3 maxP

	PxM=8x2	PxM=8x4	PxM=8x8
1. Dudek	7.78	7.92	8.09
2. davidin300	6.84	4.47	9.89
3. sylv	4.00	4.68	4.72
4. fish	11.50	2.78	12.15
5. car4	4.67	6.38	5.09
6. car11	2.17	2.36	3.57
mean	6.16	4.76	7.25

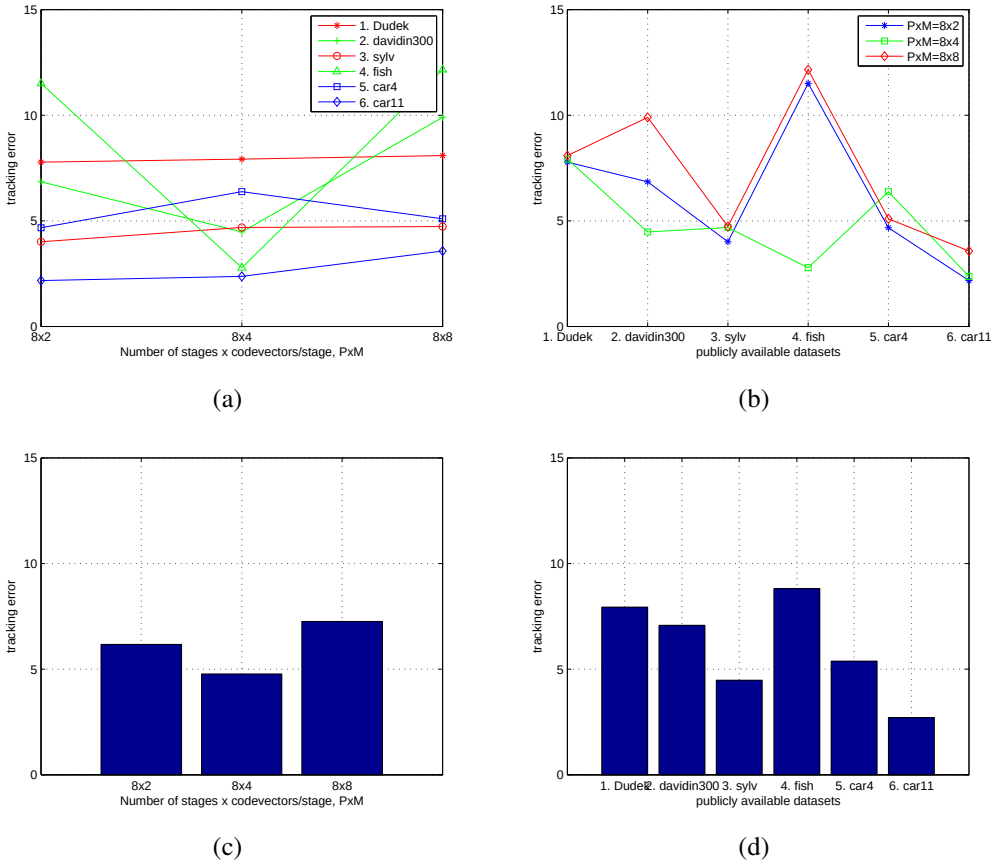


Figure 45. Tracking error for maxP based tracking for different number of codevectors per stage M with fixed stages $P = 8$ for 6 different publicly available datasets.

7.2.4 RofE

	PxM=8x2	PxM=8x4	PxM=8x8
1. Dudek	7.11	8.43	8.19
2. davidin300	9.02	6.21	5.74
3. sylv	4.12	5.54	4.83
4. fish	2.96	12.22	2.73
5. car4	4.93	5.14	5.50
6. car11	2.47	2.33	2.68
mean	5.10	6.64	4.94

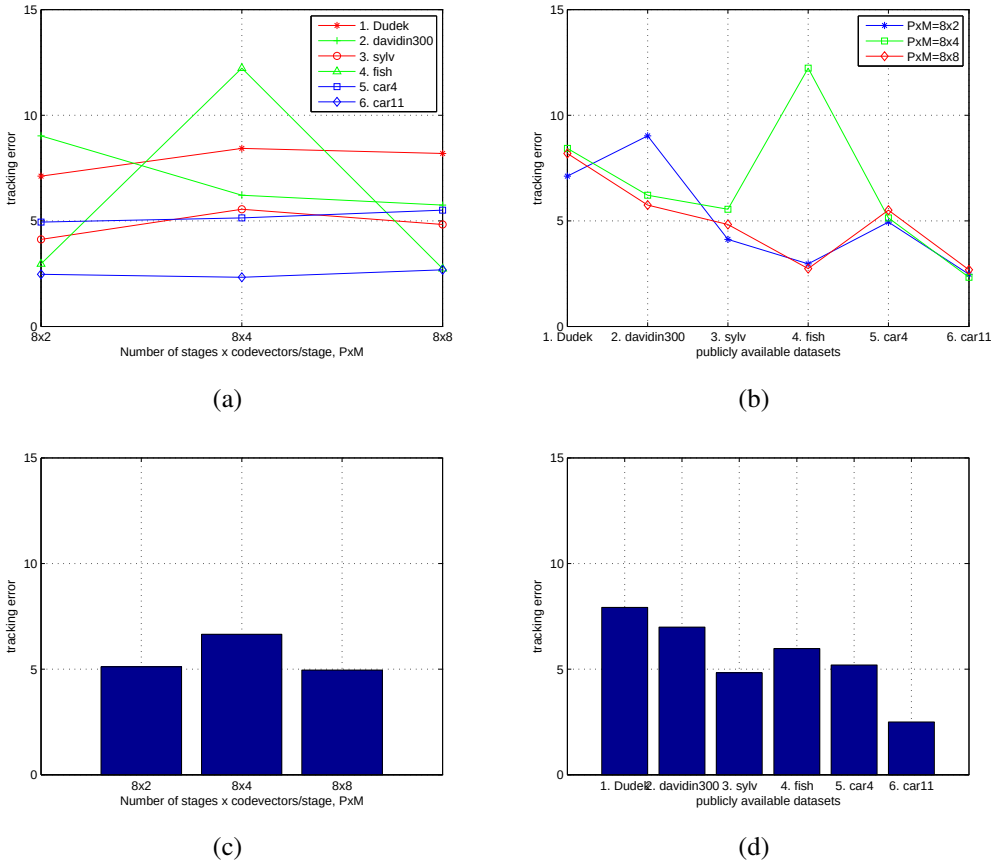


Figure 46. Tracking error for RofE based tracking for different number of codevectors per stage M with fixed stages $P = 8$ for 6 different publicly available datasets.

7.2.5 nulE

	PxM=8x2	PxM=8x4	PxM=8x8
1. Dudek	9.65	8.19	7.97
2. davidin300	7.17	5.35	4.63
3. sylv	4.81	5.74	4.74
4. fish	4.03	2.48	10.71
5. car4	5.28	5.84	6.19
6. car11	2.59	2.52	2.96
mean	5.59	5.02	6.20

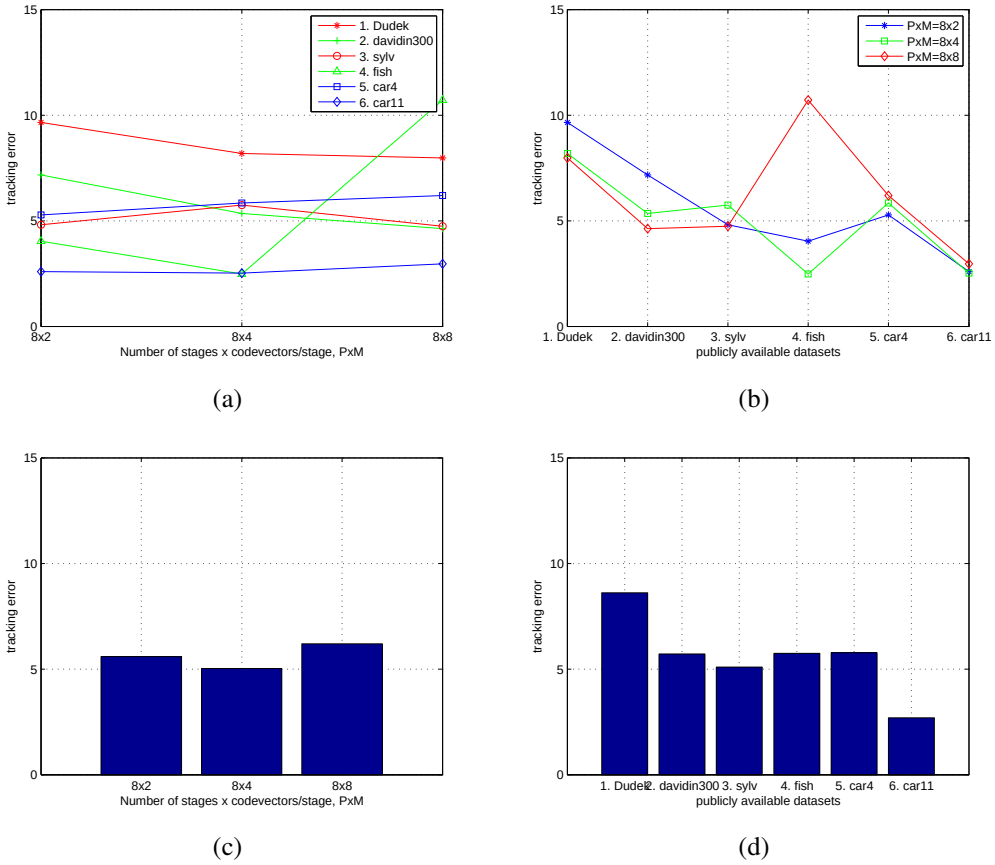


Figure 47. Tracking error for nulE based tracking for different number of codevectors per stage M with fixed stages $P = 8$ for 6 different publicly available datasets.

7.2.6 monR

	PxM=8x2	PxM=8x4	PxM=8x8
1. Dudek	11.81	9.17	8.73
2. davidin300	50.00	5.83	4.15
3. sylv	4.31	4.58	5.08
4. fish	2.89	3.62	11.94
5. car4	5.07	5.18	4.71
6. car11	2.47	2.72	2.55
mean	5.31	5.18	6.19

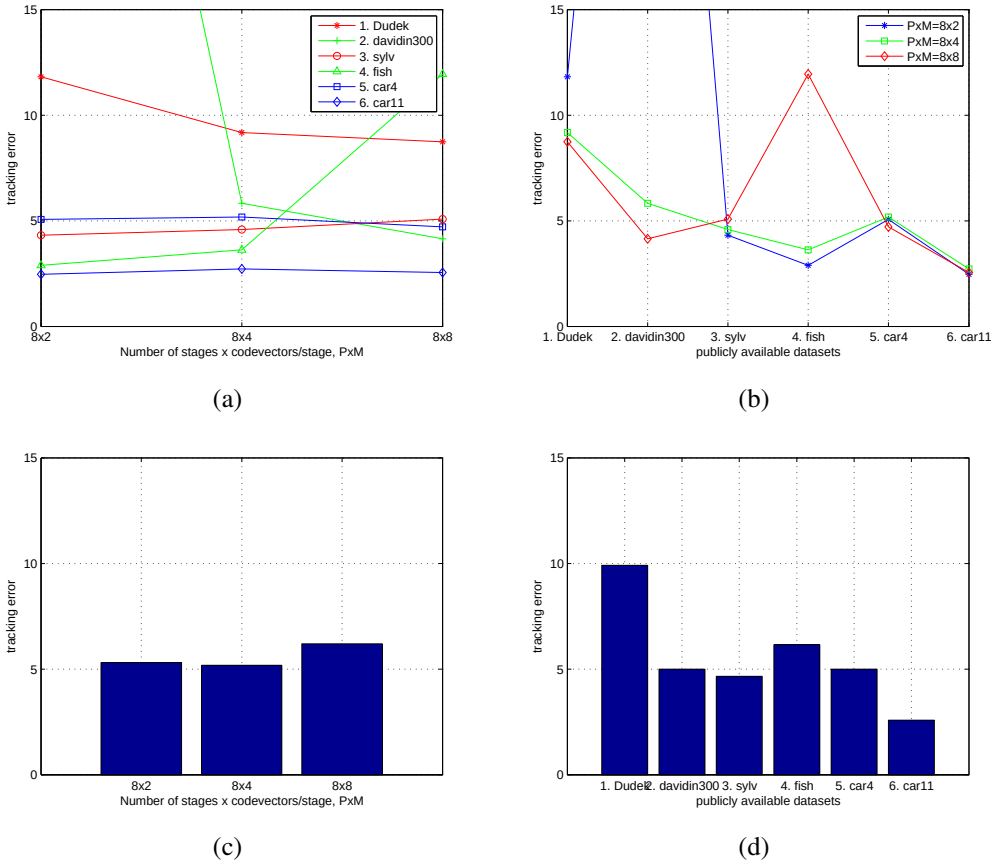


Figure 48. Tracking error for monR based tracking for different number of codevectors per stage M with fixed stages $P = 8$ for 6 different publicly available datasets. A value of 50 means that track was lost. The computation of the mean ignores the lost track case.

7.3 Example of tracking error

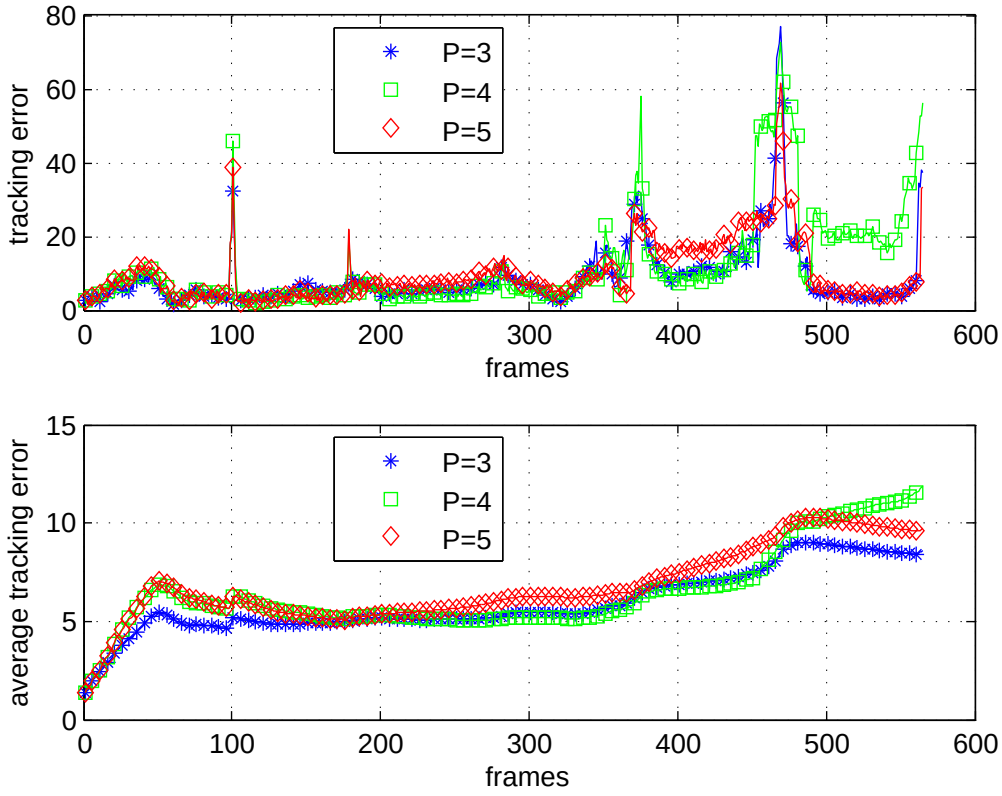
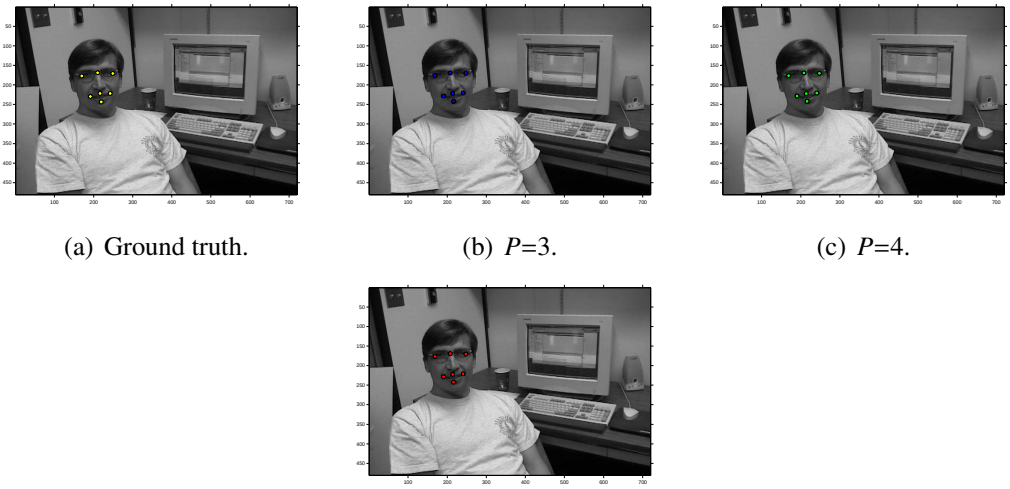


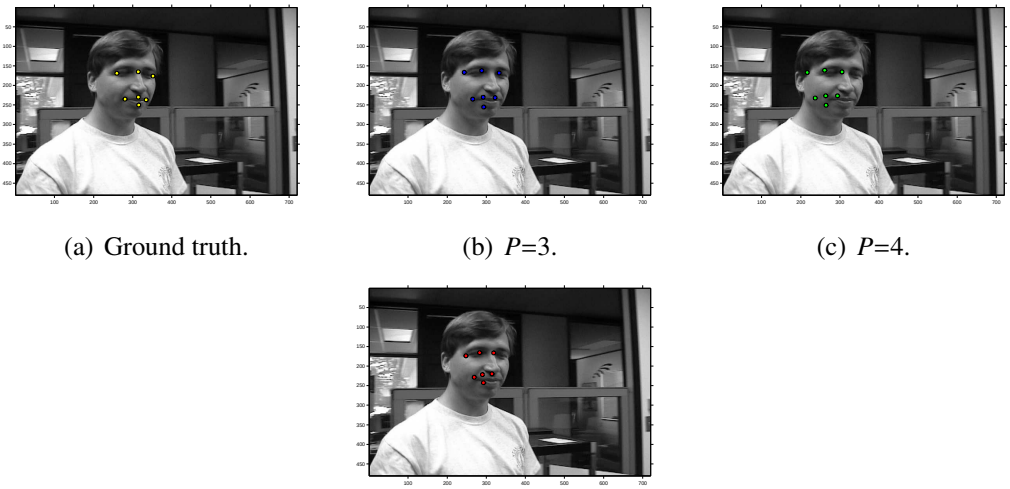
Figure 49. Tracking error for TSVQ tracker, P=3, 4, 5, Dudek sequence. The top figure shows instantaneous tracking error for the current frame while the bottom figure shows average tracking errors. Notice that at frame 457, the tracker with $P = 4$ makes an error which causes its average error to first exceed that of the P=2 tracker, and then eventually the P=5 tracker.



(a) Ground truth. (b) P=3. (c) P=4.

(d) P=5.

Figure 50. Low tracking error for TSVQ tracker, frame number=10. The overlaid circles are ground truth and estimated feature points.



(a) Ground truth. (b) P=3. (c) P=4.

(d) P=5.

Figure 51. High tracking error for TSVQ tracker, frame number=457. In this frame, the person being tracked is swiftly turning his head and moving at the same time. The tracking errors for P=3, 4 and 5 are 19.9, 47.3 and 25.1 respectively. This error causes average error for P=4 to first exceed that of the P=2 tracker, and then eventually the P=5 tracker. This shows that an error at a time when the target is undergoing large motion can be costly in the long term. This is because a wrong decision can cause the wrong snippet to be included in the training set which can cause further wrong decisions.

REFERENCES

[1] D. Ross, J. Lim, R.-S. Lin, and M.-H. Yang, "Incremental learning for robust visual tracking," *International Journal of Computer Vision*, vol. 77, pp. 125–141, 2008. 10.1007/s11263-007-0075-7.

[2] A. Yilmaz, O. Javed, and M. Shah, "Object tracking: a survey," *ACM Computing Surveys*, vol. 38, no. Copyright 2007, The Institution of Engineering and Technology, p. 45 pp., 2006.

[3] "Visualization Toolkit (VTK)." `http://www.vtk.org/`.

[4] A. Gersho and R. Gray, *Vector Quantization and Signal Compression (The Springer International Series in Engineering and Computer Science)*. Springer, 1991.

[5] C. F. Barnes, "Image-driven data mining for image content segmentation, classification, and attribution," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 45, pp. 2964–2978, 2007.

[6] S. Ullman and R. Basri, "Recognition by linear combinations of models," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 13, pp. 992 –1006, Oct. 1991.

[7] T. Breuel, "View-based recognition," *IAPR Workshop on Machine Vision Applications*, pp. 29–32, 1992.

[8] M. J. Black and A. D. Jepson, "Eigentracking: robust matching and tracking of articulated objects using a view-based representation," *International Journal of Computer Vision*, vol. 26, no. Copyright 1998, IEE, pp. 63–84, 1998.

[9] D. Skocaj and A. Leonardis, "Incremental and robust learning of subspace representations," *Image and Vision Computing*, vol. 26, no. 1, pp. 27 – 38, 2008. Cognitive Vision-Special Issue.

[10] B. Moghaddam and A. Pentland, "Probabilistic visual learning for object representation," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 19, no. 7, pp. 696–710, 1997.

[11] R. Gray, "Prehistoric cave paintings took up to 20,000 years to complete." `http://www.telegraph.co.uk/earth/3352850/Prehistoric-cave-paintings-took-up-to-20000-years-to-complete.html`, 2008.

[12] S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp, "A tutorial on particle filters for on-line non-linear/non-gaussian bayesian tracking," *IEEE Transactions on Signal Processing*, vol. 50, pp. 174–188, 2002.

[13] Y. Bar-Shalom and E. Tse, "Tracking in a cluttered environment with probabilistic data association," *Automatica*, vol. 11, no. Compendex, pp. 451–460, 1975.

[14] T. Fortmann, Y. Bar-Shalom, and M. Scheffe, "Sonar tracking of multiple targets using joint probabilistic data association," *Oceanic Engineering, IEEE Journal of*, vol. 8, pp. 173 – 184, jul 1983.

[15] D. Reid, "An algorithm for tracking multiple targets," *Automatic Control, IEEE Transactions on*, vol. 24, pp. 843 – 854, dec 1979.

[16] D. Comaniciu, "Bayesian kernel tracking," *Pattern Recognition. 24th DAGM Symposium. Proceedings (Lecture Notes in Computr Science)*, vol. 2449, pp. 438 – 45, 2002.

[17] A. Blake and M. Isard, *Active Contours: The Application of Techniques from Graphics, Vision, Control Theory and Statistics to Visual Tracking of Shapes in Motion*. Springer, 1 ed., August 2000.

[18] R. Malladi, J. Sethian, and B. Vemuri, "Shape modeling with front propagation: a level set approach," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 17, pp. 158 –175, feb 1995.

[19] I. J. Cox, "A review of statistical data association techniques for motion correspondence," *International Journal of Computer Vision*, vol. 10, no. Copyright 1993, IEE, pp. 53–66, 1993.

[20] M. Isard and A. Blake, "Condensation - conditional density propagation for visual tracking," *International Journal of Computer Vision*, vol. 29, no. Compendex, pp. 5–28, 1998.

[21] D. Sachs, S. Nasiri, and D. Goehl, "Image stabilization technology overview," *InvenSense Whitepaper*, 2006.

[22] A. Engelsberg and G. Schmidt, "A comparative review of digital image stabilising algorithms for mobile video communications," *Consumer Electronics, IEEE Transactions on*, vol. 45, no. 3, pp. 591–597, 1999.

[23] C. Stauffer and W. Grimson, "Adaptive background mixture models for real-time tracking," in *Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference on.*, vol. 2, p. 252 Vol. 2, 1999.

[24] K. Toyama, J. Krumm, B. Brumitt, and B. Meyers, "Wallflower: Principles and practice of background maintenance," in *Proceedings of the 7th IEEE International Conference on Computer Vision (ICCV'99)*, 1999.

[25] C. Harris and M. Stephens, "A combined corner and edge detector," in *4th Alvey Vision Conference*, 1988.

[26] K. Mikolajczyk and C. Schmid, "Scale & affine invariant interest point detectors," *International Journal of Computer Vision*, vol. 60, no. 1, pp. 63–86, 2004.

[27] J. Biing-Hwang and J. Gray, A., "Multiple stage vector quantization for speech coding," in *Acoustics, Speech, and Signal Processing, IEEE International Conference on ICASSP '82.*, 1982.

[28] C. Barnes, H. Fritz, and J. Yoo, "Hurricane disaster assessments with image-driven data mining in high-resolution satellite imagery," *Geoscience and Remote Sensing, IEEE Transactions on*, vol. 45, pp. 1631 –1640, june 2007.

[29] K. Sayood, *Introduction to Data Compression*. Morgan Kaufmann, 3 ed., 2005.

[30] R. Gray, "Vector quantization," *ASSP Magazine, IEEE*, vol. 1, pp. 4 – 29, apr 1984.

[31] J. Makhoul, S. Roucos, and H. Gish, "Vector quantization in speech coding," *Proceedings of the IEEE*, vol. 73, pp. 1551 – 1588, nov. 1985.

[32] M. Zaidan, C. Barnes, and S. Wicker, "Use of σ-trees as constellations in trellis-coded modulation," *Information Theory, IEEE Transactions on*, vol. 43, no. 6, pp. 2005–2012, 2002.

[33] S. Lloyd, "Least squares quantization in pcm," *Information Theory, IEEE Transactions on*, vol. 28, no. 2, pp. 129–137, 1982.

[34] J. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proc. Fifth Berkeley Symp. on Math. Statist. and Prob.*, 1967.

[35] C. Barnes and R. Frost, "Residual vector quantizers with jointly optimized code books," *Advances in Electronics and Electron Physics*, vol. 84, pp. 1–59, 1992.

[36] A. Buzo, J. Gray, A., R. Gray, and J. Markel, "Speech coding based upon vector quantization," *Acoustics, Speech and Signal Processing, IEEE Transactions on*, vol. 28, pp. 562 – 574, oct 1980.

[37] C. F. Barnes, S. A. Rizvi, and N. M. Nasrabadi, "Advances in residual vector quantization: a review," *Image Processing, IEEE Transactions on*, vol. 5, no. 2, pp. 226–262, 1996.

[38] C. F. Barnes and R. L. Frost, "Vector quantizers with direct sum codebooks," *IEEE Transactions on Information Theory*, vol. 39, no. Copyright 1993, IEE, pp. 565–80, 1993.

[39] R. Frost, C. Barnes, and F. Xu, "Design and performance of residual quantizers," in *Data Compression Conference, 1991. DCC '91.*, 1991.

[40] F. Kossentini, M. Smith, and C. Barnes, "Image coding with variable rate rvq," in *Acoustics, Speech, and Signal Processing, 1992. ICASSP-92., IEEE International Conference on*, 1992.

[41] C. Barnes, "Tree structured signal space codes," in *Coding and quantizatiion: DIMACS/IEEE Workshop, October 19-21, 1992*, p. 33, Amer Mathematical Society, 1993.

[42] F. Kossentini, M. Smith, and C. Barnes, "Necessary conditions for the optimality of variable-rate residual vector quantizers," *IEEE Transactions on Information Theory*, vol. 41, pp. 1903 –1914, 1995.

[43] C. Barnes, "A new approach to vq-based compression and classification of sensor data," in *Aerospace and Electronics Conference, 1996. NAECON 1996., Proceedings of the IEEE 1996 National*, 1996.

[44] C. Barnes, "Successive approximation source coding and image enabled data mining," in *Data Compression Conference, 2004. Proceedings. DCC 2004*, 2004.

[45] E. Trucco and K. Plakas, "Video tracking: A concise survey," *Oceanic Engineering, IEEE Journal of*, vol. 31, no. 2, pp. 520 –529, 2006.

[46] K. Cannons, "A review of visual tracking," tech. rep., Technical Report CSE-2008-07, York University, Department of Computer Science and Engineering, 2008.

[47] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer, 1st ed., October 2007.

[48] Y. Ho and R. Lee, "A bayesian approach to problems in stochastic estimation and control," *Automatic Control, IEEE Transactions on*, vol. 9, pp. 333 – 339, oct 1964.

[49] S. M. Kay, *Fundamentals of Statistical Signal Processing, Volume I: Estimation Theory (v. 1)*. Prentice Hall, 1 ed., April 1993.

[50] F. Orderud, "Comparison of kalman filter estimation approaches for state space models with nonlinear measurements," in *Scandinavian Conference on Simulation and Modeling*, 2005.

[51] S. J. Julier and J. K. Uhlmann, "A new extension of the kalman filter to nonlinear systems," in *Proc. of Aerosense: The 11th Int. Symp. on Aerospace/Defence Sensing, Simulation and Controls*, pp. 182–193, 1997.

[52] R. Van Der Merwe and E. A. Wan, "Sigma-point kalman filters for integrated navigation," in *Proceedings of the Annual Meeting - Institute of Navigation*, pp. 641 – 654, 2004.

[53] E. Wan and R. Van Der Merwe, "The unscented kalman filter for nonlinear estimation," in *Adaptive Systems for Signal Processing, Communications, and Control Symposium 2000. AS-SPCC. The IEEE 2000*, pp. 153 –158, 2000.

[54] N. Gordon, D. Salmond, and A. Smith, "Novel approach to nonlinear/non-gaussian bayesian state estimation," *Radar and Signal Processing, IEE Proceedings F*, vol. 140, pp. 107 –113, apr 1993.

[55] A. Doucet, N. Gordon, and V. Krishnamurthy, "Particle filters for state estimation of jump markov linear systems," *Signal Processing, IEEE Transactions on*, vol. 49, pp. 613 –624, mar 2001.

[56] T. Zhao and R. Nevatia, "Tracking multiple humans in crowded environment," in *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, vol. 2, pp. II–406 – II–413 Vol.2, june-2 july 2004.

[57] J. Carpenter, P. Clifford, and P. Fearnhead, "Improved particle filter for nonlinear problems," in *Radar, Sonar and Navigation, IEE Proceedings-*, vol. 146, pp. 2–7, IET, 1999.

[58] D. Crisan, P. Del Moral, and T. Lyons, "Non linear filtering using branching and interacting particle systems," *Markov Processes Related Fields*, vol. 5, no. 3, pp. 293–319, 1999.

[59] P. Del Moral and J. Jacod, *Interacting particle filtering with discrete observations*. Univ. Paul Sabatier, 1999.

[60] K. Kanazawa, D. Koller, and S. Russell, "Stochastic simulation algorithms for dynamic probabilistic networks," in *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, pp. 346–351, Citeseer, 1995.

[61] I. J. Cox and S. L. Hingorani, "An efficient implementation of reid's multiple hypothesis tracking algorithm and its evaluation for the purpose of visual tracking," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 18, no. 2, pp. 138–150, 1996.

[62] R. L. Streit and T. E. Luginbuhl, "Maximum likelihood method for probabilistic multi-hypothesis tracking," in *Signal and Data Processing of Small Targets*, 1994.

[63] Y. Bar-Shalom, F. Daum, and J. Huang, "The probabilistic data association filter," *Control Systems Magazine, IEEE*, vol. 29, pp. 82 –100, dec. 2009.

[64] C. Veenman, M. Reinders, and E. Backer, "Resolving motion correspondence for densely moving points," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 23, pp. 54 –72, jan 2001.

[65] I. K. Sethi and R. Jain, "Finding trajectories of feature points in a monocular image sequence," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. PAMI-9, no. 1, pp. 56–73, 1987.

[66] V. Salari and I. K. Sethi, "Feature point correspondence in the presence of occlusion," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 12, no. 1, pp. 87–91, 1990.

[67] K. Rangarajan and M. Shah, "Establishing motion correspondence," *CVGIP: Image Underst.*, vol. 54, no. 1, pp. 56–73, 1991.

[68] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval Research Logistics Quart. 2*, vol. Quart 2, pp. 83–97, 1955.

[69] S. Birchfield, "Elliptical head tracking using intensity gradients and color histograms," in *Computer Vision and Pattern Recognition, Proceedings. IEEE Computer Society Conference on*, pp. 232–237, 1998.

[70] A. Lipton, H. Fujiyoshi, and R. Patil, "Moving target classification and tracking from real-time video," in *Applications of Computer Vision, 1998. WACV '98. Proceedings., Fourth IEEE Workshop on*, pp. 8 –14, oct 1998.

[71] A. F. Bobick and J. W. Davis, "The recognition of human movement using temporal templates," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 23, no. 3, pp. 257–267, 2001.

[72] C. Beleznai and H. Bischof, "Fast human detection in crowded scenes by contour integration and local shape estimation," in *Computer Vision and Pattern Recognition, IEEE Conference on*, 2009.

[73] Z. Lin and L. Davis, "Shape-based human detection and segmentation via hierarchical part-template matching," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 32, pp. 604 –618, april 2010.

[74] M. Oren, C. Papageorgiou, P. Sinha, E. Osuna, and T. Poggio, "Pedestrian detection using wavelet templates," in *Computer Vision and Pattern Recognition, 1997. Proceedings., 1997 IEEE Computer Society Conference on*, pp. 193 –199, jun 1997.

[75] J. M. Rehg and T. Kanade, "Model-based tracking of self-occluding articulated objects," in *Proceedings of IEEE International Conference on Computer Vision, 20-23 June 1995*, 1995.

[76] C. F. Olson, "Maximum-likelihood template matching," in *Computer Vision and Pattern Recognition, 2000. Proceedings. IEEE Conference on*, vol. 2, 2000.

[77] S. M. Aslam, C. F. Barnes, and A. F. Bobick, "Multi target video tracking using residual vector quantization," in *International Conference on Image Processing, Computer Vision, and Pattern Recognition*, 2010.

[78] D. Comaniciu and P. Meer, "Mean shift: a robust approach toward feature space analysis," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 24, no. 5, pp. 603–619, 2002.

[79] B. Horn and B. Schunck, "Determining optical flow," *Artificial Intelligence*, vol. 17, no. 1-3, pp. 185 – 203, 1981.

[80] B. D. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision," in *Proceedings of the 7th International Joint Conference on Artificial Intelligence.*, vol. 2, pp. 674–679, 1981.

[81] J. L. Barron, D. J. Fleet, S. S. Beauchemin, and T. A. Burkitt, "Performance of optical flow techniques," *International Journal of Computer Vision*, vol. 12, no. 1, pp. 43–77, 1994.

[82] M. J. Black and P. Anandan, "The robust estimation of multiple motions: parametric and piecewise-smooth flow fields," *Computer Vision and Image Understanding*, vol. 63, no. Copyright 1996, IEE, pp. 75–104, 1996.

[83] F. P. Kuhl and C. R. Giardina, "Elliptic fourier features of a closed contour," *Computer Graphics and Image Processing*, vol. 18, no. 3, pp. 236 – 258, 1982.

[84] S. M. Aslam, C. F. Barnes, and A. F. Bobick, "Robust real time vehicle contour tracking on low quality aerial infra red imagery," in *IEEE International Geoscience and Remote Sensing Symposium (IGARSS2010)*, 2010.

[85] M. Kass, A. Witkin, and D. Terzopoulos, "Snakes: active contour models," in *Proceedings of the First International Conference on Computer Vision*, 1988.

[86] S. Niyogi and E. Adelson, "Analyzing and recognizing walking figures in xyt," in *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR '94., 1994 IEEE Computer Society Conference on*, pp. 469 –474, jun 1994.

[87] C. Rasmussen and G. D. Hager, "Probabilistic data association methods for tracking complex visual objects," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 23, no. 6, pp. 560–576, 2001.

[88] N. Peterfreund, "Robust tracking of position and velocity with kalman snakes," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 21, pp. 564 –569, jun 1999.

[89] T. Cootes, C. Taylor, D. Cooper, and J. Graham, "Active shape models-their training and application," *Computer Vision and Image Understanding*, vol. 61, no. 1, pp. 38 – 59, 1995.

[90] A. M. Baumberg and D. C. Hogg, "An efficient method for contour tracking using active shape models," in *Motion of Non-Rigid and Articulated Objects, 1994., Proceedings of the 1994 IEEE Workshop on*, pp. 194–199, 1994.

[91] A. Koschan, S. Kang, J. Paik, B. Abidi, and M. Abidi, "Color active shape models for tracking non-rigid objects," *Pattern Recognition Letters*, vol. 24, no. 11, pp. 1751 – 1765, 2003.

[92] S. S. Intille, J. W. Davis, and A. F. Bobick, "Real-time closed-world tracking," in *Computer Vision and Pattern Recognition, 1997. Proceedings., 1997 IEEE Computer Society Conference on*, pp. 697–703, 1997.

[93] C. R. Wren, A. Azarbayejani, T. Darrell, and A. P. Pentland, "Pfinder: real-time tracking of the human body," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 19, no. 7, pp. 780–785, 1997.

[94] I. Haritaoglu, D. Harwood, and L. S. Davis, "W4: real-time surveillance of people and their activities," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 22, no. 8, pp. 809–830, 2000.

[95] M. Isard and J. MacCormick, "Bramble: a bayesian multiple-blob tracker," in *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on*, vol. 2, pp. 34–41 vol.2, 2001.

[96] G. Brostow and R. Cipolla, "Unsupervised bayesian detection of independent motion in crowds," in *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, vol. 1, pp. 594 – 601, 2006.

[97] A. Jepson, D. Fleet, and T. El-Maraghi, "Robust online appearance models for visual tracking," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 25, pp. 1296 – 1311, oct. 2003.

[98] J. Krumm, S. Harris, B. Meyers, B. Brumitt, M. Hale, and S. Shafer, "Multi-camera multi-person tracking for easyliving," *vs*, p. 3, 2000.

[99] T. Darrell and A. Pentland, "Space-time gestures," in *Computer Vision and Pattern Recognition, 1993. Proceedings CVPR '93., 1993 IEEE Computer Society Conference on*, pp. 335–340, 1993.

[100] G. Hager and P. Belhumeur, "Real-time tracking of image regions with changes in geometry and illumination," in *Computer Vision and Pattern Recognition, 1996. Proceedings CVPR '96, 1996 IEEE Computer Society Conference on*, pp. 403 –410, June 1996.

[101] M. Turk and A. Pentland, "Face recognition using eigenfaces," in *Computer Vision and Pattern Recognition, 1991. Proceedings CVPR '91., IEEE Computer Society Conference on*, pp. 586 –591, June 1991.

[102] M. Bichsel and A. Pentland, "Human face recognition and the face image set's topology," *CVGIP: Image Understanding*, vol. 59, no. 2, pp. 254 – 61, 1994. face recognition;face image;geometrical transformations;topology;object recognition;physical transformations;small rotations;.

[103] "Columbia university image library (coil-100)." `http://www.cs.columbia.edu/CAVE/software/softlib/coil-100.php`.

[104] C. Qian, S. Xu, and S. Zhang, "Robust visual tracking via weighted incremental subspace learning," in *Future Computer and Communication (ICFCC), 2010 2nd International Conference on*, vol. 2, pp. V2–26 –V2–29, May 2010.

[105] Y. Li, "On incremental and robust subspace learning," *Pattern Recognition*, vol. 37, no. 7, pp. 1509 – 1518, 2004.

[106] S. Avidan, "Support vector tracking," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 26, no. 8, pp. 1064–1072, 2004.

[107] "Incremental visual tracking software." http://www.cs.toronto.edu/~dross/ivt/.

[108] D. A. Forsyth and J. Ponce, *Computer Vision: A Modern Approach*. Prentice Hall, 2002.

[109] M. Tipping and C. Bishop, "Probabilistic principal component analysis," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 61, no. 3, pp. 611–622, 1999.

[110] X. Wu, V. Kumar, J. Ross Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. McLachlan, A. Ng, B. Liu, P. Yu, *et al.*, "Top 10 algorithms in data mining," *Knowledge and Information Systems*, vol. 14, no. 1, pp. 1–37, 2008.

[111] M. Yang and Y. Wu, "Tracking non-stationary appearances and dynamic feature selection," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 2, pp. 1059–1066, IEEE, 2005.

[112] R. I. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, second ed., 2004.

[113] P. Hoff, *A first course in Bayesian statistical methods*. Springer Verlag, 2009.

[114] A. Mallet, S. Lacroix, and L. Gallo, "Position estimation in outdoor environments using pixel tracking and stereovision," in *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, vol. 4, pp. 3519–3524, IEEE, 2000.

[115] S. C. Zhu and A. Yuille, "Region competition: unifying snakes, region growing, and bayes/mdl for multiband image segmentation," *Pattern Analysis and Machine Intelligence, IEEE Transactions on DOI - 10.1109/34.537343*, vol. 18, no. 9, pp. 884–900, 1996.

[116] N. Paragios and R. Deriche, "Geodesic active regions and level set methods for supervised texture segmentation," *International Journal of Computer Vision*, vol. 46, no. 3, pp. 223 – 247, 2002.

[117] A. Elgammal, R. Duraiswami, D. Harwood, and L. S. Davis, "Background and foreground modeling using nonparametric kernel density estimation for visual surveillance," *Proceedings of the IEEE*, vol. 90, no. 7, pp. 1151–1163, 2002.

[118] P. Fieguth and D. Terzopoulos, "Color-based tracking of heads and other mobile objects at video frame rates," in *Computer Vision and Pattern Recognition. Proceedings., IEEE Computer Society Conference on*, pp. 21–27, 1997.

[119] G. Edwards, C. Taylor, and T. Cootes, "Interpreting face images using active appearance models," in *Automatic Face and Gesture Recognition, 1998. Proceedings. Third IEEE International Conference on*, pp. 300 –305, apr 1998.

[120] B. Heisele, U. Kressel, and W. Ritter, "Tracking non-rigid, moving objects based on color cluster flow," in *cvpr*, p. 257, Published by the IEEE Computer Society, 1997.

[121] V. Vapnik, *The Nature of Statistical Learning Theory (Information Science and Statistics)*. Springer, 2nd ed., November 1999.

[122] B. Karacali and H. Krim, "Fast minimization of structural risk by nearest neighbor rule," *Neural Networks, IEEE Transactions on*, vol. 14, no. 1, pp. 127–137, 2003.

[123] E. Jaynes, "On the rationale of maximum-entropy methods," *Proceedings of the IEEE*, vol. 70, pp. 939 – 952, sept. 1982.

[124] A. Leon-Garcia, *Probability and Random Processes for Electrical Engineering (2nd Edition)*. Addison-Wesley, 2 ed., August 1993.

[125] F. Escolano, P. Suau, and B. Bonev, *Information Theory in Computer Vision and Pattern Recognition*. Springer Verlag, 2009.

[126] S. Roweis and Z. Ghahramani, "A unifying review of linear gaussian models," *Neural computation*, vol. 11, no. 2, pp. 305–345, 1999.

[127] L. Sirovich and M. Kirby, "Low-dimensional procedure for the characterization of human faces," *JOSA A*, vol. 4, no. 3, pp. 519–524, 1987.

[128] P. Belhumeur, J. Hespanha, and D. Kriegman, "Eigenfaces vs. fisherfaces: recognition using class specific linear projection," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 19, pp. 711 –720, July 1997.

[129] A. Shashua, *Geometry and photometry in 3D visual recognition*. PhD thesis, Citeseer, 1992.

[130] C. Ding and X. He, "K-means clustering via principal component analysis," in *ICML '04: Proceedings of the twenty-first international conference on Machine learning*, (New York, NY, USA), p. 29, ACM, 2004.